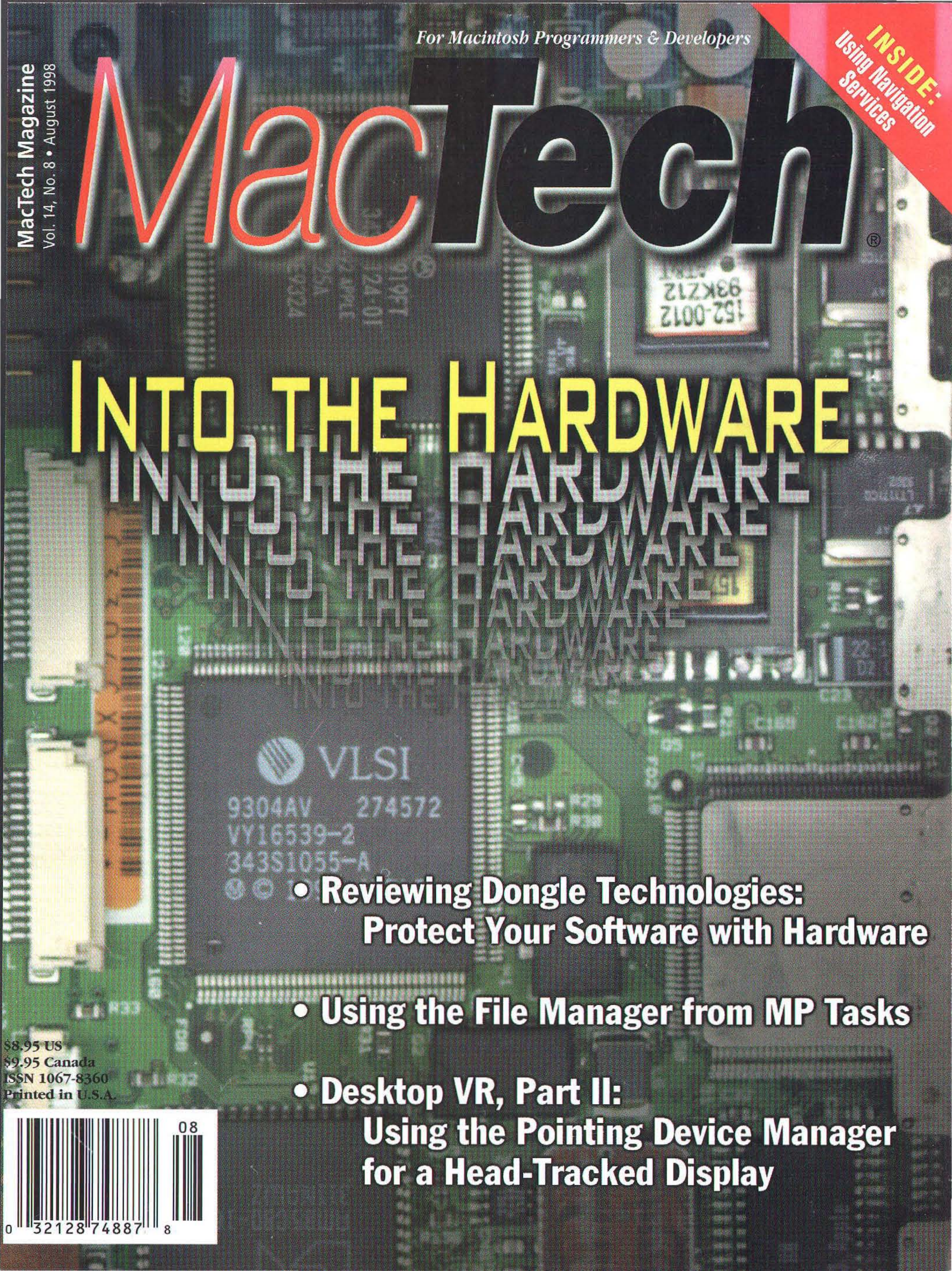
For Macintosh Programmers & Developers

MacTech®

MacTech Magazine
Vol. 14, No. 8 • August 1998

INSIDE: Using Navigation Services

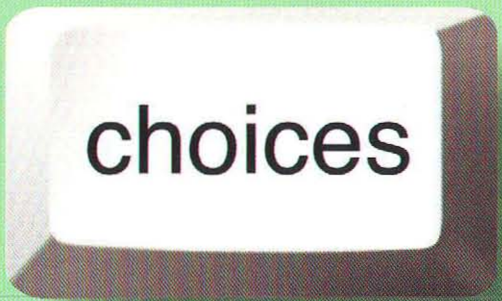


Into the Hardware

- Reviewing Dongle Technologies: Protect Your Software with Hardware
- Using the File Manager from MP Tasks
- Desktop VR, Part II: Using the Pointing Device Manager for a Head-Tracked Display

$8.95 US
$9.95 Canada
ISSN 1067-8360
Printed in U.S.A.

08
0 32128 74887 8



The industry's only provider of ADB, USB and software-based license management.

SentinelEve3-USB
All the technology and reliability of the SentinelEve3 in a USB model

It's all about choices and at Rainbow Technologies we give Macintosh software developers more. As the industry leader, Rainbow offers the most secure, reliable and technologically advanced solutions to piracy and increased sales.

SentinelEve3 — ADB-based protection
SentinelEve3-USB — ready for the new iMac
SentinelLM-Mac — a software-based license manager for standalone and network applications including Java

To learn more about the choices for secure licensing, call now or visit our website www.rainbow.com/macintosh. You'll see that for Macintosh developer solutions, there's only one choice — Rainbow.

SentinelEve3
The leading ADB-based protection solution for Macintosh developers

SentinelLM
comprehensive license management for standalone or networks

RAINBOW TECHNOLOGIES

Changing the way the world secures business.

50 Technology Drive, Irvine, CA 92618
800.705.5552 tel. 949.450.7300 fax. 949.450.7450 www.rainbow.com/macintosh

Offices and distributors located worldwide.

InstallerMaker 5.0

puts all the pieces together... even from an HTTP or FTP site.

Now with support for hierarchical packages and sublevel assignments!

- *Easiest to Use!*
- *Installation from HTTP or FTP sites*
- *Total OS 8 Compatibility*
- *Built-In Resource Compression*
- *Best Compression Available*
- *Custom Destinations*
- *Resource Installation*
- *Shrinkwrap*
- *Scripting and Recording*
- *Install or UnInstall*
- *Electronic Transaction Processing*
- *Built-In "Smart" Updaters*

There are more and more reasons StuffIt InstallerMaker 5.0 is the Complete Installation Solution. Now, your installer can automatically retrieve files from an HTTP or FTP site, so your users always get the right software. And with support for hierarchical packages and sublevel assignments, creating complex installers is a snap!

StuffIt InstallerMaker 5.0 picks up where your compiler stops. It allows you to easily create custom installers for distributing software or files. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if you're distributing on floppies) or download time (if you're distributing online).

Download a free, fully-functional copy of StuffIt InstallerMaker 5.0 from **www.aladdinsys.com**, or call (408) 761-6200 and ask for Developer Sales.

Aladdin Systems

For Macintosh Programmers & Developers

MacTech®

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MACTECH MAGAZINE

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology.

We are dedicated to the distribution of useful programming information without regard to Apple's developer status.

*For information on submitting articles, ask us for our **writer's kit** which includes the terms and conditions upon which we publish articles.*

Editorial Board of Advisors
Scott T Boyd, Jordan J. Mattson, Jim Straus, and Jon Wiederspan

Editorial Staff
Publisher • Neil Ticktin
Editor Emeritus • Eric Gundrum
Editor • Nick "nick.c" DeMello
Managing Editor • Jessica Courtney
Online Editor • Jeff Clites
Web Support • Kevin Avila

Contributing Editors
- Tantek Çelik, Microsoft Corporation
- Richard Clark
- Carl de Cordova
- Jeff Ganyard
- Jim Gochee, Connectix
- Steve Kiene, Mindvision
- Peter N Lewis
- Gary Little, Apple Computer, Inc.
- Ed Ringel
- Michael Rutman
- Rich Siegel, Bare Bones Software, Inc.

Regular Columnists
Getting Started • Dave Mark and Dan Parks Sydow
Programmer's Challenge • Bob Boonstra
From the Factory Floor • Dave Mark, Metrowerks
Tips & Tidbits • Steve Sisak
MacTech Online • Jeff Clites

XPLAIN CORPORATION
Chief Executive Officer • Neil Ticktin
President • Andrea J. Sniderman
Controller • Michael Friedman
Production Manager • Jessica Courtney
Production • Lorin Welander
Advertising Executive • Mark Altenberg
Catalog Manager • Nick DeMello
Events Manager • Susan M. Worley
Network Administrator • Chris Barrus
Customer Relations • Randy Jaramillo
Lee Ann Pham
Susan Pomrantz
Lenell Solomon
Jan Webber

Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

PRINTED WITH SOY INK

This publication is printed on paper with recycled content.

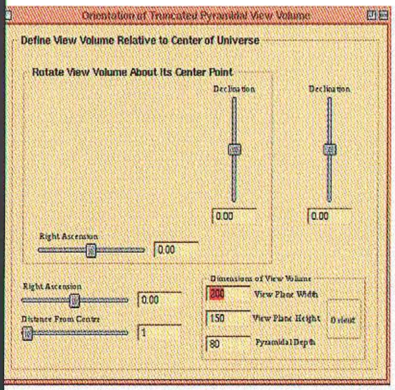
MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

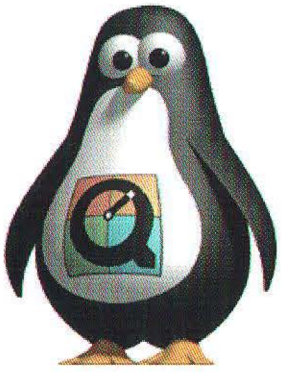
August 1998 • Volume 14, No. 8

For Macintosh Programmers & Developers
MacTech®


Orientation of Truncated Pyramidal View Volume
Define View Volume Relative to Center of Universe
Rotate View Volume About Its Center Point
Declination
Declination
Right Ascension
Right Ascension
Distance From Centre
Dimensions of View Volume
Orient

0.00 0.00 0.00 0.00 1
200 View Plane Width
150 View Plane Height
80 Pyramidal Depth

itmap Graphics page 58



Desktop VR using QuickDraw 3D, Part II page 26

About the cover...
Apple's gorgeous new PowerBook G3's have us all thinking about PowerBooks. The background of this months cover is a motherboard scan of one of MacTech's favorite PowerBooks — the Duo 230.

by Nicholas C. "nick.c" DeMello <editor@mactech.com>

Each year in June, two hundred Mac programmers converge on a single hotel in Dearborn Michigan for four days of intense coding, coffee drinking, and discussion. We call it MacHack.

The World Wide Developer Conference is Apple's show, where Apple presents their new technologies and opens a dialog with developers. The Macworld conventions are where vendors show us their new software and similarly touch base with their users. But, MacHack is something different. Suits are not worn, and no one has a briefcase. This is strictly a T-shirt and daypack crowd, and no one is trying to broadcast a message. Instead, it's simply one long party where Mac programmers the world over gather to share their opinions, experiences, and simply enjoy some time with their peers.

MacHack started twelve years ago when the University of Michigan hosted a gathering of Macintosh programmers. That was the first and last MacHack hosted by U of M. Ever since, MacHack has been run by an independent management company and organized by a committee of volunteers from the programming community.

The 1998 keynote speaker was Chris Espinosa — Apple employee number eight (and once a MacTech columnist). Hired as the eighth employee of Apple, and then working in just about every Apple department, Chris has the longest uninterrupted tenure with Apple on record. At least, until recently. It seems that when Apple acquires another company, their policy is to award its employees seniority based on their start date at the acquired company. So, when Apple acquired NeXT, all the NeXT folks were recorded on the Apple books as having started work at Apple on the dates they began with NeXT. That means, (according to Chris) the books say Steve Jobs never left Apple. That's just one of the stories we were treated to as Chris walked us through the history of Apple, from the perspective of someone who literally grew up inside the company.

MacHack is scheduled one month after WWDC. After four weeks of digesting Apple's newest strategies most programmers have some opinions, questions, and feedback on what was presented at WWDC. That makes the Apple MacHackers targets for a lot of questions and feedback. For example, Tim Holmes, Pete Gontier, and Andy Bachorski ended up in the middle of packs of folks trying to make sense out of *Carbon*, the Mac OS X schedule, and all the new technologies we're looking forward to in *Allegro*. Jordan Dea-Mattson was the first Apple employee to experience this phenomena way back at MacHack II, so he scheduled an impromptu Q&A session back then to try and field as many questions as possible. The forum was a huge success (running four hours) with folks asking Jordan about the various decisions and policies made by Apple "Why did you do .....?". Because Jordan was the only Apple representative,. everything that didn't work was, de facto, his fault. Ever since then, Apple has sent one or more Jordan-stand-ins to host an Apple feedback session (commonly referred to as the "Thank Apple Session" :-).

This year, Apple sent their VP of Software Development, Steven Glass, to host the session, and he did a great job of fielding what questions he could and taking the rest of the feedback home with him. As always, it was a great opportunity for developers to share their concerns on Apple's directions and decisions with the folks driving the bus (as well as listing the things that we need to blame Jordan for this year :-).

Apple supplied computers for the Machine room this year (which Peter N Lewis, Steve Sisak, and others put together and maintained for us). The machine room is a conference hall outfitted with computers, ethernet connections, compilers, and reference material placed at the disposal of the convention attendees. Lot's of folks were toting laptops (even a few of the extremely *kool* new PB G3's were around), but many of us don't have the resources to bring our own development environments with us — and programmers need to program. Or, to be more precise: they need to hack.

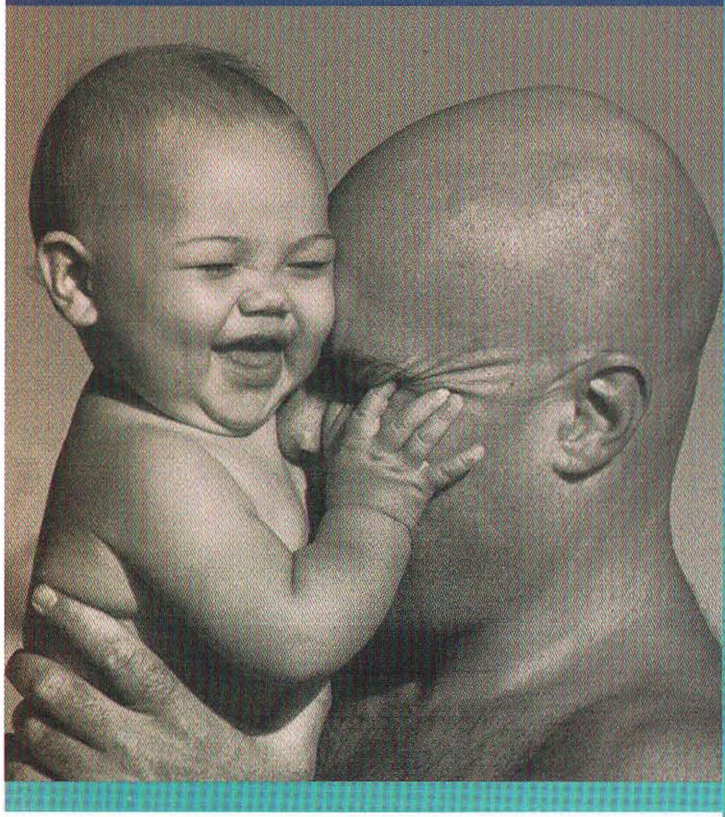
Hacks abound at MacHack. You can't put this many talented programmers in one place and not expect them to show off their wizardry. Some of the best hacks included Andy Bachorski and Nat McCully's MacsBug extension "BrickPoint" that let's you play *BreakOut* in the debugger. Ed Wynne and Matt Slot wrote a Hack that runs screen savers on your desktop (under your icons and windows). Marcus Jager and Quinn "The Eskimo!" connected two machines by a serial cable and accessed the Open Firmware bootstrap compiler on one from the other. They wrote a version of *Pong* that runs on a Mac before it boots up. All these hacks are entered into a contest at the end of the show (hosted by the *MacHax* group and run by Greg Marriot and Scott Boyd) that is the highlight of MacHack. This years winner was the asciiMac by Alexandra Ellwood and Miro Jurisic, which caused a Macintosh to render all it's graphics in colored ASCII characters — in real time!

But MacHack isn't just for the wizards among us — it's also a chance for new programmers to tap into the powerful minds and rich history of the Mac programming community. For example, Cal Simone (my personal AppleScript Guru) hosted a wonderful session exclusively for the youngest among us: "Programming for Kids". No one who attended that session was even born when MacHack I took place, but the kids loved it. Having never programmed before, the kids were fascinated by Cal's patient and step by step introduction. The half hour session turned into a 4 hour learning experience. All the kids at the session entered hacks in the juniors competition, and my guess is that these will be the future wizards of our platform.

The Mac programming community is diffuse. We need to come together periodically to share experiences and ideas, and that's really what MacHack is really about. It's a a gathering and a kind of family reunion. If you haven't been to a Hack yet, take a look at <http://www.machack.com/> and think about making your reservation for next year. See you there.

[illegible]

MacHASP® **Protects** Your Software



Your software is your baby – and you want to look after it. You created it, you developed it, you saw it right through to the moment it was ready for market. Now protect it. 50% of business software is stolen; $11 billion of developers' income is lost to piracy.* Is your software a statistic?

All over the world, more developers are protecting against piracy. They're protecting more products, on more platforms, with more security – and selling more as a result. And more of these developers are protecting with MacHASP.

To see why 25,000 developers worldwide protect with Aladdin, call and order your MacHASP Developer's Kit now!

MacHASP – The Professional Software Protection System®

- Fully licensed ADB device
- Unequalled security
- Unparalleled flexibility
- Genuine ease-of-use and transparency
- Supports System 6.05 and up, including System 8.1
- ISO 9002 quality and reliability

HASP Protects More

See Us At Comdex Spring '98 – Booth No. 1280

North America	**Aladdin Knowledge Systems Inc.** 800 223-4277, 212 564-5678, Fax: 212 564-3377, Email: hasp.sales@us.aks.com
Int'l Office	**Aladdin Knowledge Systems Ltd.** +972 3 636-2222, Fax: +972 3 537-5796, Email: hasp.sales@aks.com
Germany	**Aladdin Knowledge Systems Gmbh & Co. KG** +49 89 89 42 21-0, Fax: +49 89 89 42 21-40, Email: info@aladdin.de
UK	**Aladdin Knowledge Systems UK Ltd.** +44 1753 622266, Fax: +44 1753 622262, Email: sales@aldn.co.uk
Japan	**Aladdin Japan Co., Ltd.** +81 426 60-7191, Fax: +81 426 60-7194, Email: sales@aladdin.co.jp
France	**Aladdin France SA** +33 1 41-37-70-30, Fax: +33 1 41-37-70-39, Email: info@aladdin.fr
Benelux	**Aladdin Software Security Benelux B.V.** +31 24 648-8444, Fax: +31 24 645-1981, Email: sales@aladdin.nl
Russia	**Aladdin Software Security R.D. Ltd.** +7 095 923-0588, Fax: +7 095 928-6781, Email: aladdin@aladdin.msk.ru

Mac OS

Accelerated for Power Macintosh

IQNet Certified Quality System ISO 9002

1-800-223-4277

www.aks.com

ALADDIN®

The Professional's Choice

■ **Australia** Conlab 03 98985685 ■ **China (Beijng)** Feitian 010 62567389 **(Hong Kong)** Hastings 02 5484629 **(Shenzhen)** Hastings 0755 2328741 ■ **Czech** Atlas 02 766085 ■ **Denmark** Berendsen 039 577316 ■ **Egypt** Zeineldein 02 3604632 ■ **Finland** ID-Systems 09 8703520 ■ **Greece** Unibrain 01 6756320 ■ **India** Solution 011 2148254 ■ **Italy** Partner Data 02 26147380 ■ **Korea** Dae-A 02 8484481 ■ **Mexico** SiSoft 091 80055283 ■ **Poland** Systherm 061 480273 ■ **Portugal** Futurmatica 01 4116269 ■ **Romania** Ro Interactive 064 140383 ■ **Singapore** ITR 065 5666788 ■ **South Africa** D Le Roux 011 8864704 ■ **Spain** PC Hardware 03 4493193 ■ **Sweden** Kordab 455 307 300 ■ **Switzerland** Opag 061 7169222 ■ **Taiwan** Teco 02 5559676 ■ **Turkey** Mikrobeta 0312 4670635 ■ **Yugoslavia** Asys 021 623920

* 1997 BSA/SPA Study

by Dave Mark and Dan Parks Sydow

Window Management

How a Mac program handles multiple types of windows

Back in February we covered window-related events. In that column's WindowMaster program you saw how an application that displays a single window handles the opening, dragging, resizing, updating, and closing of that window. Those basic window-handling techniques are important because they're used in just about every Mac program — but they don't go far enough. A real-world Macintosh application usually allows for more than one window to be open at any given time. Not only that, such a program often allows more than one *type* of window to be open at a time. For instance, a drawing application might open a single window that serves as a tool palette, and then allow any number of drawing windows to also open. When such a program is running, and one of the windows needs updating, how does the program know what to draw in the window that's in need of servicing? Last month's PictLister example program presented a solution that worked for simple programs. This month we'll hold off on moving to an entirely new topic so that we can dig deeper into just how a Mac program works with windows. In this column we'll alter the PictLister code to come up with a solution that works for both simple and complex programs.

Window Management Using Global Variables

When a window-related event occurs (such as a mouse click in a window's close box), the system keeps track of the affected window. When a program calls WaitNextEvent(), the system supplies the program with a WindowPtr that references this window. Last month's program PictLister opened two windows — a list window and a picture window — and kept track of each using a global window pointer variable. When the system returned a WindowPtr to PictLister, the program compared it to it's global variables to see which window was in need of handling. Since PictLister allowed only two windows to be open, this technique was easy to implement. The simple scenario presented in PictLister is representative of some programs (a game, for instance, might display only a control window and a playing-area window). But many more applications are more complex. If a program defined a half dozen window types, and allowed any number of each to be open, you can see how difficult it would be to define a global WindowPtr variable for each, and then keep track of which windows were open throughout the running of the program. Obviously, for some applications a more sophisticated window-handling technique is called for.

Window Management Using Piggybacking

The WindowRecord data type is used to hold the information that defines a window. When a window is created from a WIND resource, that resource's information is read from disk and stored in some of the fields of a WindowRecord in memory. The first field

Dan Parks Sydow is the author of over a dozen programming books, including "Foundations of Mac Programming" by IDG Books. Dan's lending a hand on this Getting Started article.

of a WindowRecord is port, a graphics port (type GrafPort) that specifies the drawing environment of a window. The WindowPtr data type is used to provide a pointer to this first field of a WindowRecord. While your work with variables of type WindowPtr has probably lulled you into thinking of a WindowPtr variable as a pointer to a window, it is in fact simply a reference to only the drawing environment of a window. To access the other fields of a window's WindowRecord you use a variable of the WindowPeek data type. Like a variable of type WindowPtr, a variable of type WindowPeek points to the first field of a WindowRecord. Unlike the WindowPtr, though, a WindowPeek allows access to the other WindowRecord fields. **Figure 1** illustrates this (for simplicity many of the fields of the WindowRecord have been omitted).

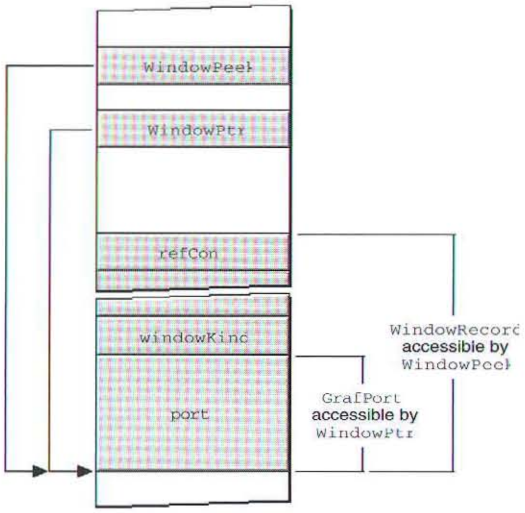

WindowPeek
WindowPtr
refCon
windowKind
port
WindowRecord accessible by WindowPeek
GrafPort accessible by WindowPtr

Figure 1. *Window access using a WindowPtr and a WindowPeek.*

Most Toolbox functions that require window access settle for a WindowPtr as a parameter. For instance, to hide a window you pass HideWindow() a pointer to the window to hide. This snippet, which creates a window from a WIND resource and then hides the window, demonstrates:

```
WindowPtrwindow;

window = GetNewWindow(kWindID, nil, kMoveToFront);
HideWindow( window );
```

For those infrequent times when you need to access a field of a window other than the port field, you need to typecast the window's WindowPtr reference to a WindowPeek. The WindowRecord field of interest is then accessed using the WindowPeek. In this next snippet the last field in a window's WindowRecord — the refCon field — is accessed in this manner. The refCon field, incidentally, is a long value that can be used to associate with a window four bytes of information.

```
WindowPeek  wPeek;
Long        windData;

wPeek = (WindowPeek)window;
windData = wPeek->refCon;
```

Now that you know some of the sordid details of how a window's information is stored and accessed, you're ready to see how to capitalize on this system. A WindowRecord holds a lot of information about a window, but it doesn't hold application-specific information. For instance, if you want your program to associate a flag with each window, there's no provision in the WindowRecord data type to allow you to store this information. For instance, consider a program that inverts the contents of some windows but not others (perhaps it's a photo darkroom utility that displays negatives in some windows). For such a program you might want to assign a Boolean value named invert to each window. To add this — or any other — information to a window you'll want to expand the WindowRecord.

To accomplish this window record expansion you define your own "peek" data type. Like the Apple-defined WindowPeek type, your data type will allow access to a WindowRecord. Your data type will, however, go on to allow access to an additional field (or fields) that you define. **Figure 2** enhances **Figure 1** to show how a variable of my own InvertWindPeek data type is used to access a window's WindowRecord and an additional field — one of type Boolean.

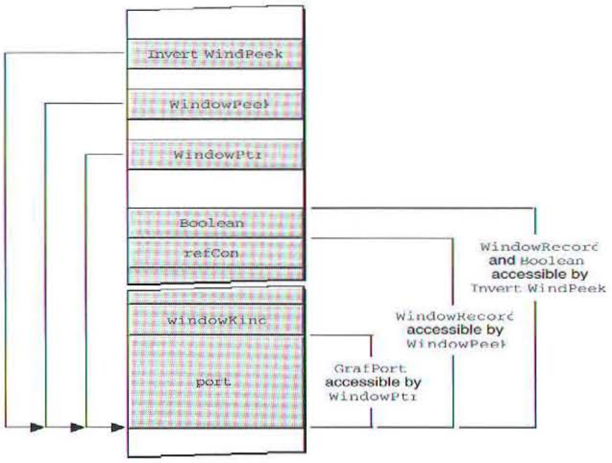

InvertWindPeek
WindowPeek
WindowPtr
Boolean
refCon
windowKind
port
WindowRecord and Boolean accessible by InvertWindPeek
WindowRecord accessible by WindowPeek
GrafPort accessible by WindowPtr

Figure 2. *Window access using a WindowPtr, WindowPeek and a InvertWindPeek.*

To create your own expanded window record data type, define a struct containing the data to be associated with a window. If you make the first field of your application-defined data type a WindowRecord, then your new data type acts much like the Apple-defined WindowPeek data type. Define an additional field in the struct for each piece of data you want associated with one of your program's windows. The effect is that your data type consists of a WindowRecord with additional data *piggybacked* onto it — thus the term *piggybacking technique*. Using piggybacking, here's how the InvertWindPeek data type discussed above might look:

```
typedef  struct
{
   WindowRecord  w;
   Boolean       invert;
} InvertWindRecord, *InvertWindPeek;
```

To make use of such an application-defined type, first allocate enough memory to hold such a structure:

```
Ptr    wStorage;

wStorage = NewPtr( sizeof( InvertWindRecord ) );
```

NewPtr() returns a generic pointer that points to a block of memory the size of one InvertWindRecord structure. This pointer can now be passed to NewWindow() to create a new window that is stored in a block of memory the size of an InvertWindRecord:

```
WindowPtr   window;

window = NewWindow( wStorage, &r, "\pUntitled", kVisible,
                    documentProc, kMoveToFront, kHasGoAway, 0L );
```

The Toolbox routine GetNewWindow()creates a window based on information stored in a WIND resource. Another Toolbox routine, NewWindow(),creates a window based on information that is instead supplied by the routine's arguments. We'll look at the arguments in more detail as we walk through the MorePictLister code.

Variable window is a WindowPtr that now points to a window that consists of a WindowRecord followed in memory by a Boolean. To gain access to the Boolean field, typecast window to an InvertWindPeek, then use the peek variable to write to or read from the invert field. Here the above-created window's invert field is assigned a value of true:

```
InvertWindPeek  invertPeek ;
Boolean         invertFlag = true;

invertPeek = (InvertWindPeek)window;
invertPeek->invert = invertFlag;
```

The piggybacking technique can be used to add any amount — and any type — of data to windows. The MorePictLister project uses this technique to add two fields to the WindowRecord that holds most of a window's data.

MorePictLister

This month's example is MorePictLister — a reworking of last month's PictLister program. MorePictLister does all the things the original program did. MorePictLister opens a list window that lists all the PICT resources available to your program. Double-clicking on a picture name in the list results in the opening of a new window that displays the selected picture. Like PictLister, MorePictLister opens a single list window. Unlike PictLister, MorePictLister displays each selected picture in its own window. **Figure 3** illustrates this.

With Allegro, or Mac OS 8.5, comes a new version of the Window Manager — version 2.0. The new Window Manager supports the wind resource — a new type of window resource that allows for better data storage than the WIND resource. In a future Getting Started we'll explore the wind resource and contrast its use with the piggybacking technique described here.

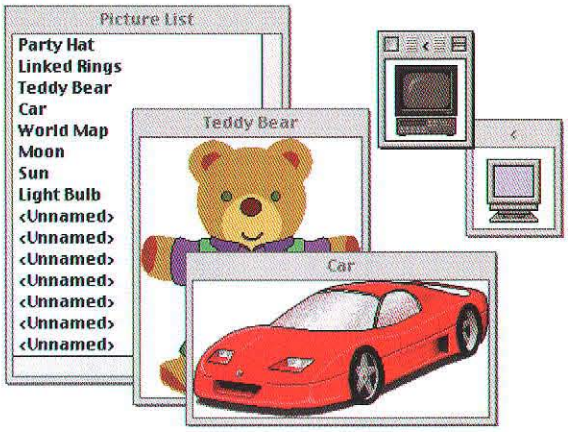


Figure 3. *Windows in the MorePictLister application.*

Creating the MorePictLister Resources

Start by creating a folder named MorePictLister inside your development folder. Launch ResEdit and create a file named MorePictLister.rsrc inside your MorePictLister folder.

MorePictLister requires the same resources that PictLister did, less the two WIND resources. There's one MBAR menu bar resource with an ID of 128, and three MENU resources with IDs of 128, 129, and 130. **Figure 4** shows the menu resources.

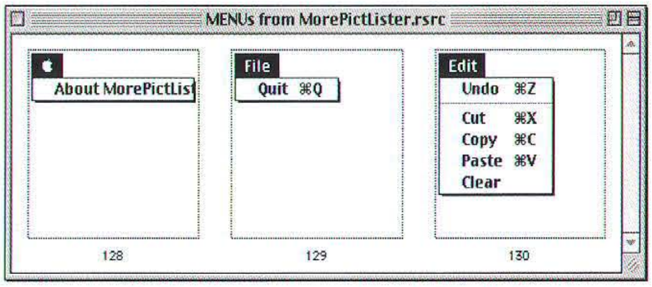


Figure 4. *The three MENU resources.*

Add to the resource file as few or as many PICT resources as you want — copying and pasting several from the Scrapbook is a quick means of getting some pictures into the file. Use ResEdit's Get Resource Info item from the Resource menu to display the Get Resource Info window for each PICT, supplying a name and checking the Purgeable check box. Make sure to save the resource file when finished.

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
*– Joe Zobkiw, author of **A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

MACUSER EDITORS' CHOICE AWARD 1993 WINNER

Accelerated for Power Macintosh

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater, 1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red, International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

Creating the MorePictLister Project

Launch CodeWarrior and create a new project based on the MacOS:C_C++:MacOS Toolbox:MacOS Toolbox Multi-Target stationary. Uncheck the Create Folder check box. Name the project MorePictLister.mcp and designate the that the project end up in the MorePictLister folder. Remove SillyBalls.c and SillyBalls.rsrc from the project; we will not be using these files in this project. From the Finder, drag and drop your MorePictLister.rsrc file into the project window. Click the OK button if you see the Add Files dialog box. This project doesn't require any of the standard ANSI libraries, so you can pare down the project file removing the ANSI Libraries folder from the project window.

Next, choose New from the File menu to create a new source code window. Save it with the name MorePictLister.c, then add the new file to the project by choosing Add Window from the Project menu. The MorePictLister source code appears in its entirety in the source code walk-through. You can type it into the MorePictLister.c file as you read, or you can save your fingers some work by simply downloading the whole project from MacTech's ftp site <ftp://ftp.mactech.com/src/>.

Walking Through the Source Code

Armed with a knowledge of the piggybacking technique and a familiarity with last month's PictLister source code, the walk-through of MorePictLister can progress swiftly. Here we'll go light on the list code (it was covered last month) and instead focus on the code that implements the piggybacking technique.

As usual, the source code listing starts off with a number of #defines — most of which come directly from last month's listing. Unfamiliar constants are discussed as they are used.

```
/******************** constants ********************/

#define kListWindow          0
#define kDAWindow            1
#define kUnknownWindow       2
#define kPictWindow          3
#define kNilWindow           4

#define kSleep               7
#define kMoveToFront         (WindowPtr)-1L
#define kListHasGoAway       false
#define kPictHasGoAway       true
#define kVisible             false

#define kListDefProc         0
#define kDrawIt              true
#define kHasGrow             true
#define kHasHScroll          true
#define kHasVScroll          true
#define kFindNext            true

#define kMinWindWidth        150
#define kMinWindHeight       60
#define kWindOrigWidth       200
#define kWindOrigHeight      255
#define kWindOrigLeft        20
#define kWindOrigTop         50
#define kBumpWindowHoriz     300
#define kBumpWindowVert      50

#define kBaseResID           128
#define kListWindID          kBaseResID
#define kPictureWindID       kBaseResID+1

#define mApple               kBaseResID
#define iAbout               1

#define mFile                kBaseResID+1
#define iQuit                1
```

PictLister uses the piggybacking technique to tie the list to the list window and to tie the PICT to the picture window. This is done by embedding a WindowRecord in each of the following typedefs.

```
typedef   struct
{
   WindowRecord      w;
   short             wType;
   ListHandle        list;
} ListWindRecord, *ListWindPeek;

typedef   struct
{
   WindowRecord      w;
   short             wType;
   short             pictResID;
} PictWindRecord, *PictWindPeek;
```

Since NewWindow() allows you to allocate your own memory for your windows, you can allocate one of the above structures instead, passing a pointer to the struct to NewWindow(). When the system provides the MorePictLister program with a WindowPtr that points to a window that needs handling, how does the program know which struct type is piggybacked on top of the window? That's what the wType field is for. When the struct is allocated, a window type is associated with it by setting the wType field to either kListWindow or kPictWindow (see the first set of #defines above). You'll see how all this works as we go along.

Last month's PictLister program required global variables to keep track of each of the two windows, the list window's list, and the picture window's picture. Thanks to piggybacking, MorePictLister can dispense with all four of those global variables. Instead, we need just a single global variable — the familiar gDone which is used to indicate when it's time to exit the main event loop.

```
/******************** globals ********************/

Boolean        gDone;
```

As always, we provide a function prototype for each function in the source file.

```
/******************** functions ********************/
void   ToolBoxInit( void );
void   MenuBarInit( void );
void   CreateListWindow( void );
void   EventLoop( void );
void   DoEvent( EventRecord *eventPtr );
void   HandleMouseDown( EventRecord *eventPtr );
short  WindowType( WindowPtr window );
void   DoContentClick( EventRecord *eventPtr,
                          WindowPtr window );
void   CreatePictureWindow( ListHandle pictList );
void   DoGrow( EventRecord *eventPtr, WindowPtr window );
void   DoUpdate( EventRecord *eventPtr );
void   DoActivate( WindowPtr window, Boolean becomingActive );
void   HandleMenuChoice( long menuChoice );
void   HandleAppleChoice( short item );
void   HandleFileChoice( short item );
```

The main() routine initializes the Toolbox, sets up the menu bar, and opens a the list window, then enters the main event loop. Recall that last month's program used main() to also open, then hide, the one picture window that was to be used to display the picture associated with a selected item in the list window. Here we forego that step. MorePictLister doesn't limit the user to displaying just one picture at a time. Instead, the program opens a new window for each selected picture.

```
/************************ main ***********************/

void main( void )
{
   ToolBoxInit();
   MenuBarInit();

   CreateListWindow();

   EventLoop();
}
```

Both ToolBoxInit() and MenuBarInit() do what's expected of them.

```
/******************** ToolBoxInit ********************/

void ToolBoxInit( void )
{
   InitGraf( &qd.thePort );
   InitFonts();
   InitWindows();
   InitMenus();
   TEInit();
   InitDialogs( nil );
   InitCursor();
}

/******************** MenuBarInit ********************/

void MenuBarInit( void )
{
   Handle        menuBar;
   MenuHandle    menu;
   menuBar = GetNewMBar( kBaseResID );
   SetMenuBar( menuBar );
   menu = GetMenuHandle( mApple );
   AppendResMenu( menu, 'DRVR' );

   DrawMenuBar();
}
```

CreateListWindow() creates the program's list window. After declaring a host of local variables, several of our #defines are used in a call to SetRect(). Setting up a rectangle that defines the size and location of the window is necessary because we haven't defined the window's look in a WIND resource, as we did last month. After setting up the rectangle, a block of memory the size of a ListWindRecord structure is reserved.

```
/****************** CreateListWindow *****************/

void CreateListWindow( void )
{
   Rect          r;
   Rect          dataBounds;
   Point         cSize, cIndex;
   short         i, dummy, numPicts;
   Handle        rHandle;
   short         resID;
   ResType       theResType;
   Str255        rName;
   WindowPtr     window;
   ListWindPeek  lwPeek ;
   ListHandle    pictList;
   Ptr           wStorage;
   SetRect( &r, kWindOrigLeft, kWindOrigTop,
            kWindOrigLeft + kWindOrigWidth,
            kWindOrigTop + kWindOrigHeight);

   wStorage = NewPtr( sizeof( ListWindRecord ) );
```

The window is created with a call to NewWindow(). Let's quickly look at the arguments passed to this Toolbox routine. The first, wStorage, is a pointer to the area in memory that will hold the window. The Rect argument r defines the initial boundaries and screen location of the window. The string is the title that's to appear in the window's title bar. kVisible is a Boolean value (defined as false) that specifies whether the window is initially visible. The Apple-defined constant documentProc specifies the look of the window (a document-style window that includes a grow box). kMoveToFront is a constant (defined to be the odd-looking value (WindowPtr)-1L) that specifies whether the window should appear in front of all other open windows. kListGoAway is a Boolean value (defined as false) that tells whether this list window should include a close box. The last argument is of type long, and is used to fill the window's refCon field with supplemental data (we're leaving this field unused, so zero is passed).

```
window = NewWindow( wStorage, &r, "\pPicture List",
                    kVisible, documentProc, kMoveToFront,
                    kListHasGoAway, 0L );
```

The new window's port is designated as the active port, and the font is set to the one that's to be used in the display of the list items.

```
SetPort( window );
TextFont( systemFont );
```

Next, we prepare to create a list one column wide and zero rows deep, with a cell size to be calculated by the List Manager, and an overall list size that is the same as the list window (less room for the list's 15-pixel wide scroll bars). This code comes directly from last months example.

```
SetRect( &dataBounds, 0, 0, 1, 0 );
SetPt( &cSize, 0, 0 );
SetRect (&r, 0, 0, kWindOrigWidth-15, kWindOrigHeight-15);
```

The list is created via a call to LNew(). Here the call is similar to that used last month. Instead of the list being returned to a global ListHandle variable, though, now we save the list to the local ListHandle variable pictList.

```
pictList = LNew( &r, &dataBounds, cSize, kListDefProc,
                 window, kDrawIt, kHasGrow, kHasHScroll,
                 kHasVScroll );
```

The selFlags field of a ListRec specifies how the list reacts to clicks in the list. Using the flag lOnlyOne tells the List Manager that only one item at a time can be highlighted in this list.

```
(**pictList).selFlags = lOnlyOne;
```

Our next step is to set the fields in our piggybacking list struct. Before we can access the struct fields we need to cast the window's reference, which is a WindowPtr, to a ListWindPeek — a pointer to a ListWindRecord. Then we'll set wType to kListWindow to mark the window as the list window, and save the list handle to the struct's list field for later recall.

```
lwPeek = (ListWindPeek)window;

lwPeek->wType = kListWindow;
lwPeek->list = pictList;
```

This next section of code adds the rows to the list. Just as we did last month, we add one row to the list for every available PICT resource. The only difference between this code and last month's is that all occurrences of the global ListHandle variable gListHandle have been changed to the local variable pictList.

```
numPicts = CountResources( 'PICT' );

for ( i = 0; i < numPicts; i++ )
{
  rHandle = GetIndResource( 'PICT', i + 1 );
  GetResInfo( rHandle, &resID, &theResType, rName );

  dummy = LAddRow( 1, i, pictList );
  SetPt( &cIndex, 0, i );

  if ( rName[ 0 ] > 0 )
    LAddToCell( &(rName[1]), rName[0], cIndex, pictList );
  else
    LAddToCell( "<Unnamed>", 10, cIndex, pictList );
}
```

Finally, the window is made visible, and LSetDrawingMode() is called to enable drawing in the list.

```
  ShowWindow( window );
  LSetDrawingMode( true, pictList );
}
```

EventLoop() and DoEvent() remain unchanged from last month.

```
/******************** EventLoop ********************/

void EventLoop( void )
{
  EventRecord   event;

  gDone = false;
  while ( gDone == false )
  {
    if ( WaitNextEvent( everyEvent, &event, kSleep, NULL ) )
      DoEvent( &event );
  }
}

/******************** DoEvent ********************/

void DoEvent( EventRecord *eventPtr )
{
  char theChar;
  Boolean  becomingActive;

  switch ( eventPtr->what )
  {
    case mouseDown:
      HandleMouseDown( eventPtr );
      break;
    case keyDown:
    case autoKey:
      theChar = eventPtr->message & charCodeMask;
      if ( (eventPtr->modifiers & cmdKey) != 0 )
        HandleMenuChoice( MenuKey( theChar ) );
      break;
    case updateEvt:
      DoUpdate( eventPtr );
      break;
    case activateEvt:
      becomingActive = ( (eventPtr->modifiers & activeFlag)
                          == activeFlag );
      DoActivate( (WindowPtr)eventPtr->message,
                  becomingActive );
      break;
  }
}
```

HandleMouseDown() has just one change from last month's version. Take a look at the code under the inGoAway case label. A pointer to the window that's in need of handling is a part of the EventRecord that the system sent to the program. Instead of comparing this window pointer to a global list window pointer, as last month's program did, here we call our own WindowType() function to see if the window to close is the list window. WindowType() is described next.

```
/****************** HandleMouseDown ****************/

void HandleMouseDown( EventRecord *eventPtr )
{
  WindowPtrwindow;
  short       thePart;
  long        menuChoice;

  thePart = FindWindow( eventPtr->where, &window );

  switch ( thePart )
  {
    case inMenuBar:
      menuChoice = MenuSelect( eventPtr->where );
      HandleMenuChoice( menuChoice );
      break;
    case inSysWindow :
      SystemClick( eventPtr, window );
      break;
    case inContent:
      DoContentClick( eventPtr, window );
      break;
    case inGrow:
      DoGrow( eventPtr, window );
      break;
    case inDrag :
      DragWindow( window, eventPtr->where,
                  &qd.screenBits.bounds );
      break;
    case inGoAway:
      if ( TrackGoAway( window, eventPtr->where ) )
      {
        if ( WindowType( window ) == kPictWindow )
          CloseWindow( window );
      }
      break;
  }
}
```

WindowType() is called anytime our MorePictLister program wants to identify a window's type; pass WindowType() a WindowPtr, and the function returns one the application-defined constants that are used to categorize the window. If the window has a negative windowKind field, it's a Desk Accessory (by Apple's own definition). If the window's wType field is kListWindow or kPictWindow, one of those constants is returned, else kUnknownWindow is returned.

```
/******************** WindowType ******************/

short  WindowType( WindowPtr window )
{
  if ( window == nil )
    return( kNilWindow );
  if ( ((WindowPeek)window)->windowKind < 0 )
    return( kDAWindow );
  if ( ((ListWindPeek)window)->wType == kListWindow )
    return( kListWindow );

  if ( ((ListWindPeek)window)->wType == kPictWindow )
    return( kPictWindow );

  return( kUnknownWindow );
}
```

DoContentClick() is called when the mouse is clicked in the specified window's content region. The functionality of the routine is the same as last month's version, though the implementation has changed slightly. Here we no longer use the global list handle variable. Instead, we peek in the list window's struct to get the list handle that's stored with the window. Next, a call to LClick() is made. This routine handles all types of clicks, from clicks in the scroll bars to clicks in the list cells. LClick() returns true if a double-click occurs. In that case, we'll invoke the application-defined routine SetWindowPict() to determine which picture is to be displayed in the picture window.

```
/****************** DoContentClick *****************/

void DoContentClick( EventRecord *eventPtr, WindowPtr window )
{
  ListHandle    pictList;
  ListWindPeek  lwPeek;

  if ( window != FrontWindow() )
  {
    SelectWindow( window );
  }
  else if ( WindowType( window ) == kListWindow )
  {
    SetPort( window );

    GlobalToLocal( &(eventPtr->where) );

    lwPeek = (ListWindPeek)window;
    pictList = lwPeek->list;

    if ( LClick( eventPtr->where, eventPtr->modifiers,
                 pictList ) )
      CreatePictureWindow( pictList );
  }
}
```

In last month's example, a double-click on a list item resulted in a call to an application-defined function named SetWindowPicture(). There we displayed the appropriate picture in the program's one picture window. The piggybacking technique makes it easy to keep track of multiple windows, so in this version of the program we handle a list item selection by opening a new picture window; the user can have any number of pictures displayed at once. CreatePictureWindow() takes care of the work of opening a picture window and assigning a picture to that window. First, a number of variables are declared.

```
/**************** CreatePictureWindow ***************/

void CreatePictureWindow( ListHandle pictList )
{
  Cell          cell;
  Handle        rHandle;
  Rect          r;
  short         resID;
  ResType       theResType;
```

Multi-Platform Environment

· Port Your Mac Code To Windows!

· Code Once, Use Twice!

· Develop On Two Platforms!

Cheaply. Efficiently. Swiftly.

http://www.9003inc.com

PORT YOUR MAC CODE TO WINDOWS!

Code once, use twice!

With the Multi-Platform Environment (MPE) as a porting and development platform developers can:

- **Create products for Mac & Win operating systems!**
- **Fast conversion from Mac to Win!**
- **Reduced overhead - only one team of engineers required!**
- **Reduced time and resource costs!**
- **Easy development of future additions/upgrades and bug-fixes!**
- **Low maintenance cost - only one set of code to maintain!**

Code or port your application once with the MPE, then compile anytime to get an application for both Macintosh and Windows 95/NT platforms.

The MPE consists of a programming library, set of tools and documentation that permits development to proceed as you would on the Mac, building Mac and Win apps, with Mac and Win binary compatible data files for true cross-platform data.

Utilizing the MPE, developers can easily create applications for both Mac and Win platforms, and quickly reach the Windows market with their existing Mac apps!

Be sure to visit our online Porting Oasis for free porting information!

http://www.9003inc.com

9003inc. Canada Corp. 296 Richmond Street West, Ste. 400, Toronto, Ontario, Canada M5V 1X2
Phone: (416) 343-0032   Fax: (416) 343-4976   Email: mpe@9003inc.com

9003inc.

```
Str255        rName;
WindowPtr   window;
PictWindPeek  pwPeek;
PicHandle   pic;
Ptr           wStorage;
```

CreatePictureWindow() performs many of the chores that last month's SetWindowPicture() handled. The function moves to the first cell in the list, then calls LGetSelect() to move through the list to find the selected cell and put the coordinates of that cell in variable cell. If a highlighted cell is found, we use cell.v to retrieve the appropriate PICT resource. We'll be basing the size and location of the picture window on the size of the picture, so here we store the size of the picture in Rect variable r. The left and top coordinates of r are then shifted right kBumpWindowHoriz pixels and down kBumpWindowVert pixels. That leaves the overall size of the rectangle unaffected, but guarantees that the top-left corner of the soon-to-be-created picture window won't end up in the very upper-left corner of the screen where it would be partly obscured by the menu bar. Next, a call to GetResInfo() is made to obtain the PICT resource's name.

```
  SetPt( &cell, 0, 0 );

  if ( LGetSelect( kFindNext, &cell, pictList ) )
  {
    rHandle = GetIndResource( 'PICT', cell.v + 1 );
    pic = (PicHandle)rHandle;

    r = (**pic).picFrame;
    OffsetRect( &r, kBumpWindowHoriz, kBumpWindowVert );

    GetResInfo( rHandle, &resID, &theResType, rName );
```

Next, memory for a PictWindRecord is allocated and the new storage is used to create the new picture window.

```
    wStorage = NewPtr( sizeof( PictWindRecord ) );
    window = NewWindow( wStorage, &r, rName, kVisible,
                        noGrowDocProc, kMoveToFront,
                        kPictHasGoAway, 0L );
```

The new window's title was set in the call to NewWindow(), but now we'll check to see if the PICT resource was named. If it wasn't, then we make that fact known by using the string "<Unnamed>" for the window's title. At this point the window is all set up, so we display it and make it active.

```
    if ( rName[ 0 ] == 0 )
      SetWTitle( window, "\p<Unnamed>" );
    ShowWindow( window );
    SelectWindow( window );
```

Finally, the PictWindRecord's wType field is set to kPictWindow and the PICT's resource ID is stored in the PictResID field for use when updating the window (as described just ahead in DoUpdate()).

```
    pwPeek = (PictWindPeek)window;
    pwPeek->wType = kPictWindow;
    pwPeek->pictResID = resID;
  }
}
```

DoGrow() is called when the mouse is clicked in a window's grow box. The routine begins by calling WindowType() to determine what kind of window is to be resized (picture window's can't be resized, but here we're allowing for a future enhancement that may allow them to be). If the window is a list window, we first establish the minimum and maximum size of the window. Next, we call GrowWindow(). If the window was resized, we call SizeWindow() and LSize() to resize the window and to let the List Manager know that the list has been resized.

```
/********************** DoGrow *********************/

void DoGrow( EventRecord *eventPtr, WindowPtr window )
{
  Rect        r;
  Cell        cSize;
  long        windSize;
  ListHandle  pictList;

  if ( WindowType( window ) == kListWindow )
  {
    r.top = kMinWindHeight;
    r.bottom = 32767;
    r.left = kMinWindWidth;
    r.right = 32767;

    windSize = GrowWindow( window, eventPtr->where, &r );
    if ( windSize )
    {
      SetPort( window );
      EraseRect( &window->portRect );

      SizeWindow( window,
           LoWord (windSize),
           HiWord(windSize), true );

      pictList = ((ListWindPeek)window)->list;
      LSize( LoWord(windSize)-15,
             HiWord(windSize)-15, pictList );
```

Next, local variable cSize is set to the current cell size in. The horizontal part of cSize is then used to change the cells width to match the new width of the resized window. A call to LCellSize() resizes all the cells, and a call to InvalRect() forces an update.

```
      HLock( (Handle)pictList );
      cSize = (*pictList)->cellSize;
      HUnlock( (Handle)pictList );

      cSize.h = LoWord( windSize ) - 15;
      LCellSize( cSize, pictList );
      InvalRect( &window->portRect );
    }
  }
}
```

Back in DoEvent() you saw that DoUpdate() is called to handle an update event. This version of DoUpdate() is similar to last month's. The primary difference is that now list access is carried out by accessing the list field of the list window's ListWindRecord structure. DoUpdate() begins by retrieving the WindowPtr from the EventRecord, setting the port, and then calling BeginUpdate().

```
/********************* DoUpdate ********************/

void DoUpdate( EventRecord *eventPtr )
{
  WindowPtr    window;
  Rect           r;
  ListWindPeek  lwPeek;
  ListHandle    pictList;
```

```
PictWindPeek  pwPeek;
PicHandle   pic;

window = (WindowPtr)(eventPtr->message);
SetPort( window );
BeginUpdate( window );
```

If the window is the list window, we redraw the grow box, gain access to the window's list, then call LUpdate() to let the List Manager update the list.

```
if ( WindowType( window ) == kListWindow )
{
  DrawGrowIcon( window );

  lwPeek = (ListWindPeek)window;
  pictList = lwPeek->list;
  LUpdate( (**pictList).port->visRgn, pictList );
}
```

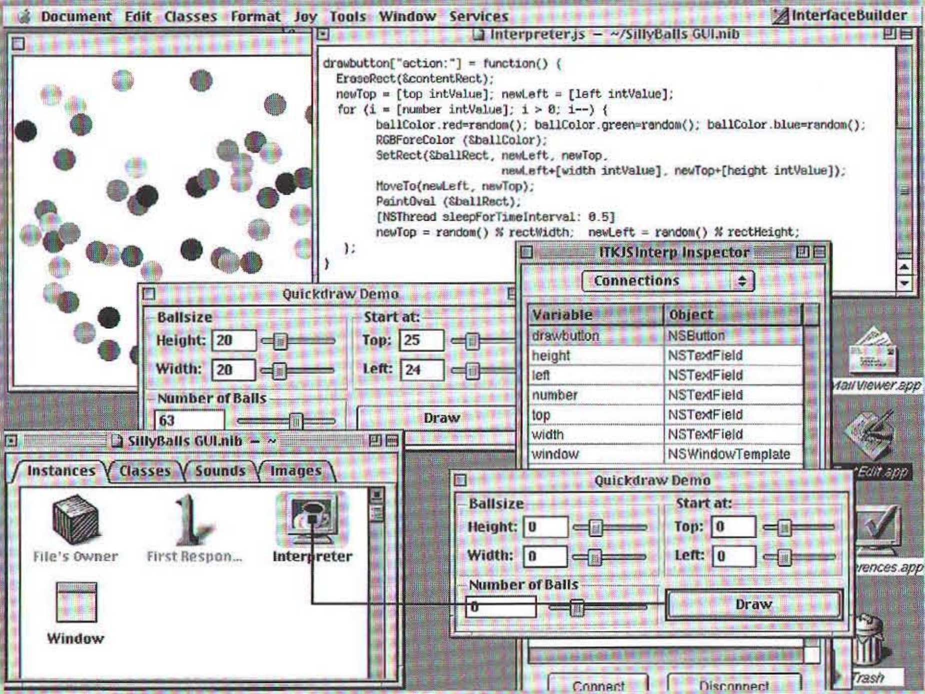
If the window is the picture window, we determine the size of the window's picture by looking at the size of picture window's port. In CreateWindowPicture() we based the size of the picture window on the size of the picture that was to be displayed in it. The picture resource ID is held in the pictResID field of the picture window's PictWindRecord, so we pull the ID from that field, use it in a call to GetPicture() to load the corresponding PICT resource into memory, and then call DrawPicture() to do the drawing. A call to EndUpdate() tells the program that updating is finished.

```
  else if ( WindowType( window )  == kPictWindow )
  {
    r = window->portRect;

    pwPeek = (PictWindPeek)window;

    pic = GetPicture( pwPeek->pictResID );

    DrawPicture( pic, &r );
  }

  EndUpdate( window );
}
```

DoActivate() handles activate events. The picture window doesn't need any special activate event processing, so this routine focuses on the list window.

```
/******************** DoActivate ********************/

void DoActivate( WindowPtr window, Boolean becomingActive )
{
  ListWindPeek  lwPeek;
  ListHandle    pictList;

  if ( WindowType( window ) == kListWindow )
  {
    lwPeek = (ListWindPeek)window;
    pictList = lwPeek->list;

    if ( becomingActive )
      LActivate( true, pictList );
    else
```

Java-Script with Joy!



```
drawbutton["action:"] = function() {
  EraseRect(&contentRect);
  newTop = [top intValue]; newLeft = [left intValue];
  for (i = [number intValue]; i > 0; i--) {
        ballColor.red=random(); ballColor.green=random(); ballColor.blue=random();
        RGBForeColor (&ballColor);
        SetRect(&ballRect, newLeft, newTop,
                        newLeft+[width intValue], newTop+[height intValue]);
        MoveTo(newLeft, newTop);
        PaintOval (&ballRect);
        [NSThread sleepForTimeInterval: 0.5]
        newTop = random() % rectWidth;  newLeft = random() % rectHeight;
  };
}
```

Variable	Object
drawbutton	NSButton
height	NSTextField
left	NSTextField
number	NSTextField
top	NSTextField
width	NSTextField
window	NSWindowTemplate

Use the Joy-enhanced Interface Builder application to create applications within minutes. Graphically link interface elements to JavaScript variables. Add scripts to the '.nib' file. Use standard C, Objective-C, and Java syntax from within Joy's JavaScript. Call any API. Save your interface file as a stand-alone application. Test your application immediately. No compiling.   No linking.   No need to relaunch the application.

www.aaa-plus.com

t r i p l e   a   p l u s

Joy Developer 2.0

by AAA+ Software

Java-Script Carbon and Yellow Box!

- Access any C, Objective-C, or Java API from the new JavaScript-compliant scripting language.
- Test your code immediately with the Joy-enhanced InterfaceBuilder.
- Develop full-featured applications in minutes. No compiling. No linking.
- Browse *any* application's objects and classes at runtime and send them messages.

Joy 2.0 ships in Q3/98 as a FREE upgrade to registered users. Prerelease versions are available on request.

Joy 1.0 received a MacTech Best Tool for New Technologies Finalist Award.

Download and test Joy today:
http://www.aaa-plus.com/joy/

Price: $ 399

Available for Rhapsody PPC, Intel, Windows NT/95, and OPENSTEP.

Email your order today:
order@aaa-plus.com

Finalist
Macworld 1997
Editors' Choice Awards

```
      LActivate( false, pictList );

    DrawGrowIcon( window );
  }
}
```

The remaining menu-handling code is identical to last months code — so we don't need to comment on the **HandleMenuChoice()**, **HandleAppleChoice()**, or **HandleFileChoice()** routines.

```
/***************** HandleMenuChoice *****************/

void HandleMenuChoice( long menuChoice )
{
  short  menu;
  short  item;

  if ( menuChoice != 0 )
  {
    menu = HiWord( menuChoice );
    item = LoWord( menuChoice );

    switch ( menu )
    {
      case mApple:
        HandleAppleChoice( item );
        break;
      case mFile:
        HandleFileChoice( item );
        break;
    }
    HiliteMenu( 0 );
  }
}

/***************** HandleAppleChoice *****************/

void HandleAppleChoice( short item )
{
  MenuHandle  appleMenu;
  Str255      accName;
  short       accNumber;

  switch ( item )
  {
    case iAbout:
      SysBeep( 10 );
      break;
    default:
      appleMenu = GetMenuHandle( mApple );
      GetMenuItemText( appleMenu, item, accName );
      accNumber = OpenDeskAcc( accName );
      break;
  }
}

/***************** HandleFileChoice *****************/

void HandleFileChoice( short item )
{
  switch ( item )
  {
    case iQuit:
      gDone = true;
      break;
  }
}
```

Running MorePictLister

Select Run from the Project menu to run MorePictLister. Like last month's program, MorePictLister displays a menu bar featuring the , File, and Edit menus, and the Picture Lister window. Double-click on an item in the list of the list window and a new window that displays the selected picture appears. Choosing another list item opens still another window.

Till Next Month

Next month we'll take a look at Apple events. Apple defines four specific event types that it refers to as the four required Apple events. To date our relatively simple projects haven't included support for Apple events, so it's time to get with the program! By supporting Apple events, you can make your own Mac application more "Finder-friendly." For instance, if your application is running when the user chooses the Shut Down command from the Finder's Special menu, you'll want the operating system to be able to quit your application along with all other running programs. Apple events make this possible. Until then, experiment with this month's piggybacking technique to create a program that opens and keeps track of all manner of windows. See you next month...

MT

MkLinux

Microkernel Linux for the Power Macintosh

MkLinux is a complete system, based on Linux 2 and the Mach 3 microkernel. It includes development tools (gcc, gdb, perl, ...), X11R6.3, and hundreds of other commands.

MkLinux builds and runs most Intel-based Linux software. It works efficiently and reliably on a wide range of Power Macintosh platforms. Visit Apple Computer's MkLinux web site (www.mklinux.apple.com) and Prime Time Freeware's web site (www.ptf.com) to find out more about MkLinux!

MacPerl: Power and Ease

MacPerl is a robust and powerful port of Perl 5 to the Mac. It runs under the Finder and MPW, supports Apple Events and Toolbox calls, and is generally quite nifty! For details on MacPerl and associated products (book, CD, etc.), visit the MacPerl Pages (www.ptf.com/macperl).

Prime Time Freeware	info@ptf.com
370 Altair Way, #150	(408) 433-9662
Sunnyvale, CA 94086	(408) 433-0727 fax

Have You Made a Visit to the New MacTech® Web Site?

http://www.mactech.com

Here are some highlights:

Macintosh Development News

Keep yourself informed. The site has all the current breaking news in the development community and even archives past news pieces.

Search the Site

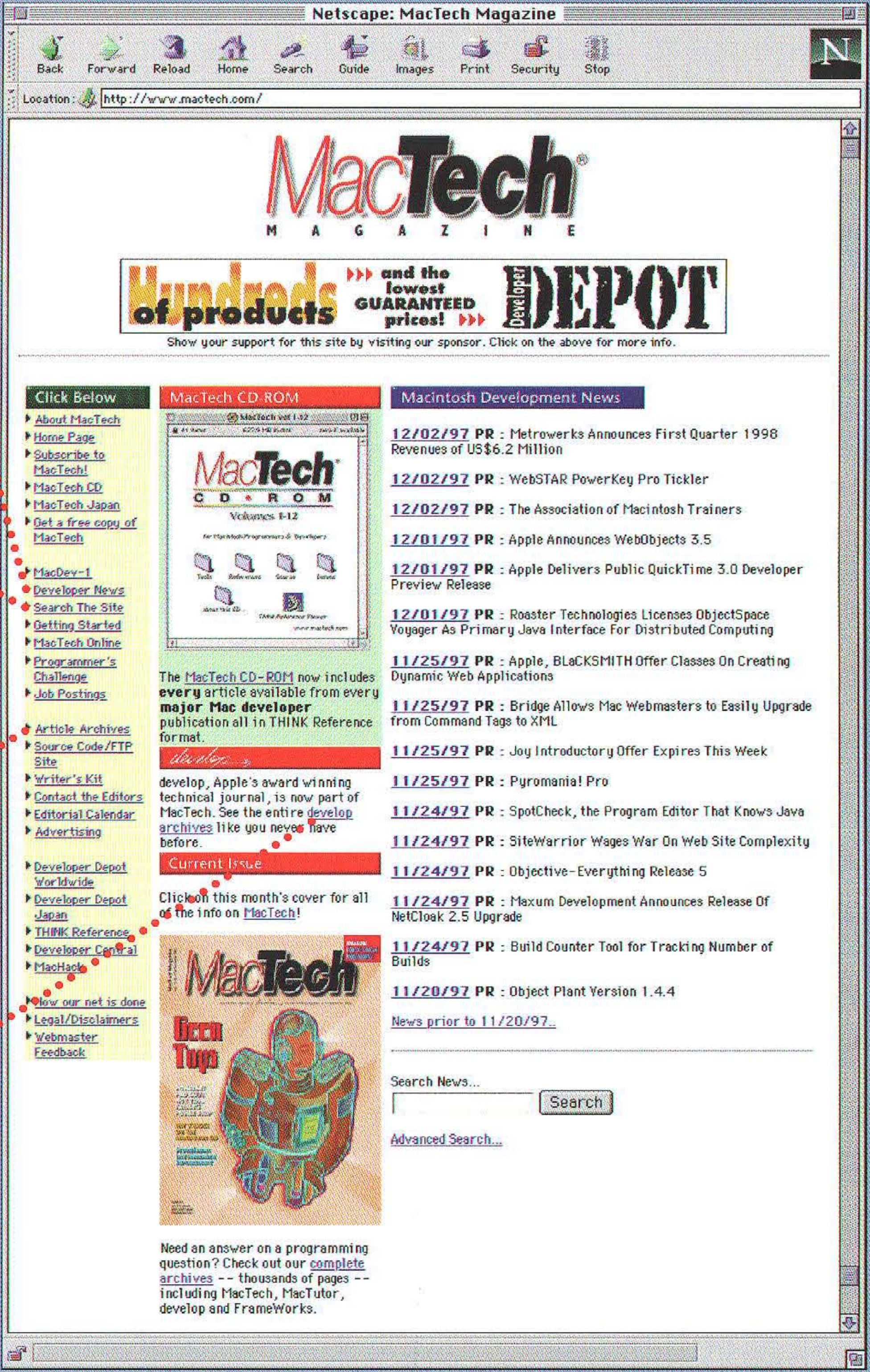
Search like never before. Search and find that article, news piece or source code you're looking for via the new Phantom search engine courtesy of Maxum.

Article Archives

And you can have it all. These archives have thousands of pages of content in them from the histories of MacTech, MacTutor, develop, and FrameWorks magazines.

develop

develop. Need we say more. The entire history of develop, Apple's award winning technical journal. That is 29 issues from 1990 – 1997.


Netscape: MacTech Magazine
Back
Forward
Reload
Home
Search
Guide
Images
Print
Security
Stop
Location: http://www.mactech.com/
MacTech MAGAZINE
Hundreds of products and the lowest GUARANTEED prices! Developer DEPOT

Show your support for this site by visiting our sponsor. Click on the above for more info.

Click Below

- About MacTech
- Home Page
- Subscribe to MacTech!
- MacTech CD
- MacTech Japan
- Get a free copy of MacTech
- MacDev-1
- Developer News
- Search The Site
- Getting Started
- MacTech Online
- Programmer's Challenge
- Job Postings
- Article Archives
- Source Code/FTP Site
- Writer's Kit
- Contact the Editors
- Editorial Calendar
- Advertising
- Developer Depot Worldwide
- Developer Depot Japan
- THINK Reference
- Developer Central
- MacHack
- How our net is done
- Legal/Disclaimers
- Webmaster Feedback

MacTech CD-ROM

The MacTech CD-ROM now includes **every** article available from every **major Mac developer** publication all in THINK Reference format.

develop

develop, Apple's award winning technical journal, is now part of MacTech. See the entire develop archives like you never have before.

Current Issue

Click on this month's cover for all of the info on MacTech!

Need an answer on a programming question? Check out our complete archives -- thousands of pages -- including MacTech, MacTutor, develop and FrameWorks.

Macintosh Development News

12/02/97 **PR** : Metrowerks Announces First Quarter 1998 Revenues of US$6.2 Million

12/02/97 **PR** : WebSTAR PowerKey Pro Tickler

12/02/97 **PR** : The Association of Macintosh Trainers

12/01/97 **PR** : Apple Announces WebObjects 3.5

12/01/97 **PR** : Apple Delivers Public QuickTime 3.0 Developer Preview Release

12/01/97 **PR** : Roaster Technologies Licenses ObjectSpace Voyager As Primary Java Interface For Distributed Computing

11/25/97 **PR** : Apple, BLaCKSMITH Offer Classes On Creating Dynamic Web Applications

11/25/97 **PR** : Bridge Allows Mac Webmasters to Easily Upgrade from Command Tags to XML

11/25/97 **PR** : Joy Introductory Offer Expires This Week

11/25/97 **PR** : Pyromania! Pro

11/24/97 **PR** : SpotCheck, the Program Editor That Knows Java

11/24/97 **PR** : SiteWarrior Wages War On Web Site Complexity

11/24/97 **PR** : Objective-Everything Release 5

11/24/97 **PR** : Maxum Development Announces Release Of NetCloak 2.5 Upgrade

11/24/97 **PR** : Build Counter Tool for Tracking Number of Builds

11/20/97 **PR** : Object Plant Version 1.4.4

News prior to 11/20/97...

Search News...

Search

Advanced Search...

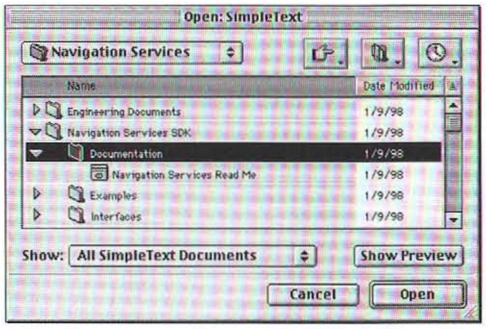
by Keith Mortensen
Edited by the MacTech Magazine Editorial Staff

Using Navigation Services

Every Macintosh programmer knows you can't write an application without going through the experience of using the StandardFile package. One discovers almost immediately the limitations and constraints of StandardFile, and begins to realize the standard dialogs don't support the features most developers need. Thus, adding your own features by customizing them is the only alternative.

If you have had to customize any of the StandardFile dialogs, can you remember your first dialog hook function? Have you wondered what "pseudo items" meant? Do you remember "copying" the System file's 'DLOG' and 'DITL' resources (ID -6042) to make your very own open dialog? How about the feelings of uncertainty when "changing" these resources? Have you resorted to writing a filter callback procedure just because the SFTypeList holds no more than 4 file types?

Over the years your applications have changed, but those StandardFile dialogs haven't. I'm glad to say those days are over. In this article we will introduce the tools of Apple's successor to the StandardFile package — Navigation Services.


Open: SimpleText

Navigation Services

Name	Date Modified
Engineering Documents	1/9/98
Navigation Services SDK	1/9/98
Documentation	1/9/98
Navigation Services Read Me	1/9/98
Examples	1/9/98
Interfaces	1/9/98

Show: All SimpleText Documents

Figure 1. *The Open Dialog.*

Meet Navigation Services, a new set of tools for document management. It hosts a suite of "utility dialogs" for opening and saving documents, choosing volumes, choosing and creating folders. More additional standard alerts are also provided to free you from having to design your own.

Navigation Services provides a set of navigation tools for the user, including:

- Shortcuts – for quick access to mounted volumes and other desktop items.
- Favorites – for easy navigation to frequently used items.
- Recent – to provide a history of the users work.
- Browser List – a hierarchical Finder-like view of the file system.

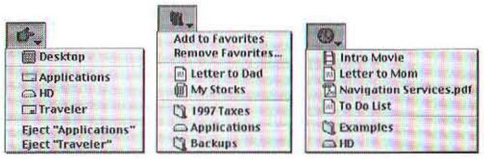


Figure 2. *Shortcuts, Favorites and Recent menus.*

Keith Mortensen (morten.k@apple.com) is a software engineer for Apple Computer working on various human interface products for the Mac OS, currently contributing to Navigation Services. When Keith is off work, he's busy programming, working on his Suburban, and spends his spare time four wheeling and enjoying the outdoors at the Russian River and Tuolumne County areas of California.

There are several areas where Navigation Services provides "memory assistance" to improve the user experience. Each time you open a document, it will remember: your last location, what you last opened, where the dialog box was displayed on your screen, the dimensions of the dialog box, and the browser's sort key and sort order. It also remembers the selection for every folder or container you visit. More importantly, this persistence is on a per-application and per-dialog basis. Thus, an Open dialog's position and size for a word processor application may be different than of a spreadsheet application.

Navigation Services even gives you automatic built-in translation of your documents. If SimpleText were to use Navigation Services, it would be capable of opening any document that can be translated into text, picture, or movie formats. Lastly, it also gives you the capabilities to add your own interface to these utility dialogs through a well-defined system in tune with the over-all user interface.

A Navigation Services "Savvy" Application

We will be going through the process of making your application Navigation Services "savvy" by studying some simple code examples. They are designed to make the transition quick and painless. You needn't worry about how your application is built. The interface can be accessed using 68K, CFM-68K and PowerPC code. This gives you the flexibility to adapt and remain compatible to a wider variety of systems. By calling NavServicesAvailable, you can determine if Navigation Services is installed and ready to run.

NavGetFile

The first call I recommend using is NavGetFile, a fully featured "get" dialog of Navigation Services. Here you will find most of the features you need. The following example shows how to use NavGetFile for opening multiple documents.

```
OSErr OpenDocument( )
{
  OSErr                 theErr = noErr;
  NavReplyRecord        theReply;
  NavDialogOptions      dialogOptions;
  long                  count;
  NavTypeListHandle  openTypeList = NULL;
  NavEventUPP        eventProc =
                        NewNavEventProc( myEventProc );
  NavObjectFilterUPP filterProc =
                        NewNavObjectFilterProc( myFilterProc );

  // default behavior for browser and dialog:
  theErr = NavGetDefaultDialogOptions( &dialogOptions );

  // setup our NavTypeList for filtering:
  openTypeList = (NavTypeListHandle)GetResource('open', 128);

  theErr = NavGetFile(  NULL, &theReply, &dialogOptions,
                        eventProc, NULL, filterProc,
                        openTypeList, NULL );
  if ( theReply.validRecord && theErr == noErr )
  {
    FSSpec      finalFSSpec;
    AEDesc      resultDesc;
    AEKeywordkeyWord;
    if ( theErr == noErr )
    {
      // we are ready to open the document(s), grab
      // information about each file for opening:
      if ((theErr = AECountItems( &(theReply.selection),
                                   &count )) == noErr)
        for (long index=1;index<=count;index++)
          if ((theErr = AEGetNthDesc( &(theReply.selection),
              index,typeFSS,&keyWord,&resultDesc)) == noErr)
          {
            BlockMoveData( *resultDesc.dataHandle,
                           &finalFSSpec, sizeof(FSSpec) );
            if ((theErr = DoOpenFile(&finalFSSpec))== noErr)
            theErr = AEDisposeDesc( &resultDesc );
          }
    }
    theErr = NavDisposeReply( &theReply );
  }

  DisposeRoutineDescriptor( filterProc );
  DisposeRoutineDescriptor( eventProc );
  if (openTypeList != NULL)
    ReleaseResource( (Handle)openTypeList );
  return theErr;
}
```

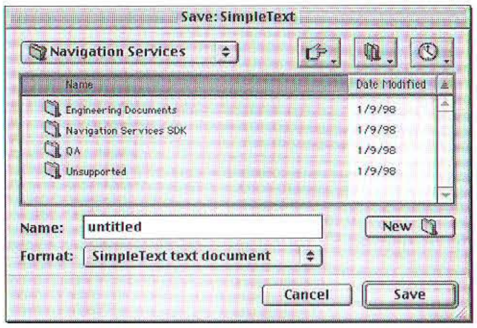
Opening Documents with AppleEvents

Navigation Services returns in the selection field of the NavReplyRecord an AEDescList or AppleEvent Descriptor List. It contains one or more items to be opened. To open the documents, one nifty technique is to send to your application an 'odoc' AppleEvent with the items found in this list. This method saves you from having to deal with AppleEvent Descriptors. Use your AppleEvent handler to perform the open, as if you received the AppleEvent from the Finder.

```
theErr = SendOpenAE(theReply.selection);

static OSStatus SendOpenAE(AEDescList list)
{
  OSStatus        err;
  AEAddressDesc theAddress;
  AppleEvent      dummyReply, theEvent;

  theAddress.descriptorType= typeNull;
  theAddress.dataHandle  = NULL;

  do {
    ProcessSerialNumber psn;

    err = GetCurrentProcess(&psn);
    if ( err != noErr) break;

    err = AECreateDesc(typeProcessSerialNumber, &psn,
                sizeof(ProcessSerialNumber), &theAddress);
    if ( err != noErr) break;

    dummyReply.descriptorType= typeNull;
    dummyReply.dataHandle  = NULL;

    err = AECreateAppleEvent(kCoreEventClass,
                kAEOpenDocuments, &theAddress,
              kAutoGenerateReturnID, kAnyTransactionID,
                &theEvent);
    if ( err != noErr) break;

    err = AEPutParamDesc(&theEvent, keyDirectObject, &list);
    if ( err != noErr) break;

    err = AESend(&theEvent, &dummyReply, kAEWaitReply,
              kAENormalPriority, kAEDefaultTimeout,
              NULL, NULL);
    if ( err != noErr) break;
  } while (false);

  return err;
}
```

NavPutFile

Programming the save dialog is similar to using NavGetFile with two additions: you provide a "suggested" saved file name and file format. Once you are done with NavPutFile, call NavCompleteSave to perform translation of the saved document in case the user chose a different save format than your own. It is important you call NavCompleteSave, as this call will provide more "save" features in future versions. The next example shows how to use NavPutFile. No object filtering is necessary and the Favorites and Recent menus will show only containers.



Name	Date Modified
Engineering Documents	1/9/98
Navigation Services SDK	1/9/98
QA	1/9/98
Unsupported	1/9/98

Name: untitled
Format: SimpleText text document

Figure 3. *The Save Dialog.*

```
OSErr SaveDocument(WindowPtr theDocument)
{
  OSErr               theErr = noErr;
  NavReplyRecord      theReply;
  NavDialogOptionsdialogOptions;
  OSType              fileTypeToSave = 'TEXT';
  OSType              creatorType = 'xAPP';
  NavEventUPP       eventProc = NewNavEventProc(myEventProc);

  // default behavior for browser and dialog:
  theErr = NavGetDefaultDialogOptions( &dialogOptions );

  GetWTitle( theDocument, dialogOptions.savedFileName );

  theErr = NavPutFile( NULL, &theReply, &dialogOptions,
            eventProc, fileTypeToSave, creatorType, NULL );

  if (theReply.validRecord && theErr == noErr)
  {
    FSSpec      finalFSSpec;
    AEDesc      resultDesc;
    AEKeyword keyWord;

    // retrieve the returned selection:
    if (( theErr = AEGetNthDesc( &(theReply.selection), 1,
              typeFSS, &keyWord, &resultDesc )) == noErr )
    {
      BlockMoveData( *resultDesc.dataHandle, &finalFSSpec,
                      sizeof(FSSpec) );
      if (theReply.replacing)
        theErr = FSpDelete( &finalFSSpec );
      if ( theErr == noErr )
        if (( theErr = FSpCreate( &finalFSSpec, creatorType,
              fileTypeToSave, smSystemScript )) == noErr )
          if (( theErr = WriteNewFile(&finalFSSpec))==noErr)
            theErr = NavCompleteSave( &theReply,
                                    kNavTranslateInPlace);
      AEDisposeDesc( &resultDesc );
    }
    theErr = NavDisposeReply( &theReply );
  }
  DisposeRoutineDescriptor( eventProc );
  return theErr;
}
```

Dialog and Browser Options

A wide variety of features and options to change the look and behavior of the interface are available by setting up a NavDialogOptions structure:

```
struct NavDialogOptions {
  UInt16    version;
  NavDialogOptionFlags    dialogOptionFlags; // dialog/browser options
  Point     location;             // the dialog's screen coordinates
  Str255    clientName;           // your program's name
  Str255    windowTitle;          // a complete window title
  Str255    actionButtonLabel;    // the action button's text
  Str255    cancelButtonLabel;    // the cancel button's text
  Str255    savedFileName;        // for NavPutFile, the default file name
  Str255    message;              // the banner or prompt message string
  UInt32    preferenceKey;  // unique value used to map to set of preferences
  Handle    popupExtension;       // client's added popupMenu items
};
typedef struct NavDialogOptions NavDialogOptions;

enum {
  kNavDefaultNavDlogOptions= 0x000000E4,
                          // the default features
  kNavNoTypePopup = 0x00000001,
                          // turn off using the type popup menu
  kNavDontAutoTranslate  = 0x00000002,
                          // do not automatically translate documents
  kNavDontAddTranslateItems= 0x00000004,
                          // add translated types to type popup menu
  kNavAllFilesInPopup  = 0x00000010,
                          // add "All Documents" to type popup
  kNavAllowStationery = 0x00000020,
                          // add Stationery menu option to type popup
  kNavAllowPreviews = 0x00000040,
                          // use the Previews option for documents
  kNavAllowMultipleFiles = 0x00000080,
                          // browser selects and returns multiple objects
  kNavAllowInvisibleFiles = 0x00000100,
                          // browser showss invisible objects
  kNavDontResolveAliases = 0x00000200,
                          // don't resolve aliases, return the alise file
  kNavSelectDefaultLocation = 0x00000400,
                          // select the default location
  kNavSelectAllReadableItem = 0x00000800
                          // select "All Readable Documents"

};
typedef UInt32 NavDialogOptionFlags;
```

You can easily setup this structure to use the default settings by calling NavGetDefaultDialogOptions. It is better to call this routine anyway since it initializes the structure automatically for you. Then you can turn on or off particular features by adding or subtracting option flags.

Filtering Objects

Navigation Services does "object filtering" to determine what objects to display in the Recent Documents list, Favorites list, and the browser list.

Your application has two ways of filtering objects. One method is to use a filter callback procedure. A simpler way is to use a NavTypeList structure. This data structure is identical to the 'open' resource format defined by the Translation Manager. It contains the clients creator, and a list of OSTypes which represent the file types your application can open.

By using a filter callback procedure and/or NavTypeList structure, clients can filter out unwanted file objects from the interface. You can directly filter any object, including files, folders, volumes, aliases, etc. You can use both the NavTypeList and filter callback procedure if you wish. Both are designed to

work together. For example, if the NavTypeList contains TEXT and PICT types, only TEXT and PICT files will be passed into your filter callback procedure.

```
pascal Boolean myFilterProc( AEDesc* theItem, void* info,
                             NavCallBackUserData callBackUD,
                             NavFilterModes filterMode)
{
  OSErr    theErr = noErr;
  Boolean  display = true;
  NavFileOrFolderInfo* theInfo = (NavFileOrFolderInfo*)info;

  if ( theItem->descriptorType == typeFSS )
    if ( !theInfo->isFolder )
      if ( theInfo->
          fileAndFolder.fileInfo.finderInfo.fdType!='TEXT')
        display = false;
  return display;
}
```

Handling Events

An event callback routine is used by your application to respond to events. Events can be Mac OS or Navigation Services events. By providing an event callback routine, you can keep your application's windows updated, handle idle events as well as communicate and interact with Navigation Services. For example, to respond to operating system events check for the kNavCBEvent message. This is one of a set of useful event messages defined in the API. Each message yields different data used for a particular event. The following shows the NavCBRec structure used in handling events:

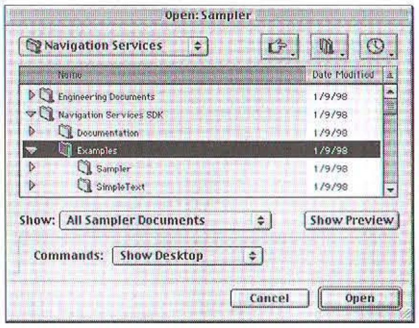
```
struct NavCBRec {
  UInt16      version;
  NavContext  context;          // used to refer to Navigation Services
  WindowPtr   window;           // the Navigation dialog
  Rect        customRect;       // coordinate rectangle of customization area
  Rect        previewRect;      // coordinate rectangle of the preview area
  NavEventDataInfo  eventData;  // event info to examine/interpret events
  char        reserved[226];
};

pascal void myEventProc(   NavEventCallbackMessage
                             callBackSelector,
                           NavCBRecPtr callBackParms,
                             NavCallBackUserData callBackUD )
{
  WindowPtr pWindow = (WindowPtr)callBackParms->
                                 eventData.event->message;
  myApp* curApp = (myApp*)callBackUD;    // context to ourselves

  switch ( callBackSelector )
  {
    case kNavCBEvent:
      switch ( callBackParms->eventData.event->what )
      {
      case updateEvt:
        curApp->DoUpdate(pWindow,
              (EventRecord*)callBackParms->eventData.event);
        break;
      }
      break;
  }
}
```

Customizing the Dialogs

Throughout the years, StandardFile has been heavily customized in almost every piece of software. As a result, the user experience is often inconsistent and confusing. Up until now, most of us have learned the behaviors of StandardFile, and we all have become used to them. As a result, Apple gets stuck with the difficulties of trying to advance the human interface without breaking current applications.

Navigation Services provides a number of calls and options which address needs in the past that required customization of the StandardFile dialogs. By using the features of this API, you will not have to customize the dialogs in a manner that will cause incompatibilities with future versions of Navigation Services. The need to change the Navigation Services dialogs, however, is greatly reduced.

The example in **Figure 4** is how Sampler, an example application included with the Navigation Services SDK, customizes NavGetFile. It adds a "Commands" popup menu control. Sampler is responsible for allocating any controls and responding to mouseDown and keyDown events.



Name	Date Modified
Engineering Documents	1/9/98
Navigation Services SDK	1/9/98
Documentation	1/9/98
Examples	1/9/98
Sampler	1/9/98
SimpleText	1/9/98

Figure 4. *An example of dialog customization.*

Dialog Space Negotiations

If you need to add your own interface you will need to negotiate a customization space by responding to this kNavCBCustomize event. The callBackParms->customRect field describes the custom area in local coordinates anchored to the lower left of the dialog's type popup or its browser. If you want to customize the dialog, you must complete the dimensions of the rectangular area by responding to kNavCBCustomize in your event callback procedure. Complete the dimensions in callBackParms->customRect for your custom area.

The following example is in response to kNavCBCustomize and shows you how to tell Navigation Services the space you need. Agreeing upon the customized space is like a round of negotiations between you and Navigation Services, coming up with an amount of space both of you can agree upon. Once you are satisfied with the given size, exit your event callback procedure with the same custom area. This will tell Navigation Services that an agreement has been made. The minimum guaranteed space is 402 pixels wide by 40 pixels high. The next example shows you how to complete the area for customization:

```
{
// here are the desired dimensions for our custom area:
short neededWidth =
        callBackParms->customRect.left + kCustomWidth;
short neededHeight =
        callBackParms->customRect.top + kCustomHeight;
// check to see if this is the first round of negotiations:
if (( callBackParms->customRect.right == 0) &&
      (callBackParms->customRect.bottom == 0 ))
{
   // it is, so tell NavServices what dimensions we want:
   callBackParms->customRect.right = neededWidth;
   callBackParms->customRect.bottom = neededHeight;
}
else
{
   // we are in the middle of negotiating:
   if ( globals->gLastTryWidth !=
         callBackParms->customRect.right )
      if ( callBackParms->customRect.right < neededWidth )
         // is the given width too small for us?
         callBackParms->customRect.right = neededWidth;

   // is the given height too small for us?
   if ( globals->gLastTryHeight !=
         callBackParms->customRect.bottom )
      if ( callBackParms->customRect.bottom < neededHeight )
         callBackParms->customRect.bottom = neededHeight;
}

// remember our last size so the next time we can re-negotiate:
globals->gLastTryWidth = callBackParms->customRect.right;
globals->gLastTryHeight = callBackParms->customRect.bottom;
}
```

Sending "Nav" Commands

If you supply an event callback routine, you can in turn "call back" to Navigation Services to control the dialog and its browser. You can do this by calling NavCustomControl. NavCustomControl accepts a "context" or reference to Navigation Services which is provided by the NavCBRec, when the client's event routine is called. Its selector field determines the type of control call being made, and a generic pointer is used to convey parameter data. Using a set of custom control messages you can control the dialog and browser in various ways. Below are some examples of how to use NavCustomControl:

```
Str255 fileName;
theErr = NavCustomControl(callBackParms>context,
                          kNavCtlGetEditFileName,&fileName);

Str255 newFolderName = "\puntitled folder";
theErr = NavCustomControl(callBackParms->context,
                          kNavCtlNewFolder,&newFolderName);

theErr = NavCustomControl(callBackParms->context,
                          kNavCtlShowDesktop,NULL);
```

The following is a list of commands or NavCustomControlMessages you can send to Navigation Services. These commands can be used in your event or preview callback procedures, where appropriate. The arrows illustrate how data is passed to and from NavCustomControl.

```
enum {
   kNavCtlShowDesktop= 0,       // show desktop, parms = nil
   kNavCtlSortBy = 1,           // sort key field, parms->NavSortKeyField
   kNavCtlSortOrder = 2,        // sort order,parms->NavSortOrder
   kNavCtlScrollHome   = 3,     // scroll list home, parms = nil
   kNavCtlScrollEnd = 4,        // scroll list end, parms = nil
   kNavCtlPageUp = 5,           // page list up, parms = nil
   kNavCtlPageDown = 6,         // page list down, parms = nil
   kNavCtlGetLocation = 7,      // get current location, parms<-AEDesc
   kNavCtlSetLocation = 8,      // set current location, parms->AEDesc
   kNavCtlGetSelection  = 9,    // get current selection, parms<-AEDescList
   kNavCtlSetSelection  = 10,   // set current selection, parms->AEDescList
   kNavCtlShowSelection = 11,   // make selection visible, parms = nil
   kNavCtlOpenSelection = 12,   // open view of selection, parms = nil
   kNavCtlEjectVolume = 13,     // eject volume, parms->vRefNum
   kNavCtlNewFolder = 14,          // create a new folder, parms->StringPtr
   kNavCtlCancel = 15,             // cancel dialog, parms = nil
   kNavCtlAccept = 16,             // accept dialog default, parms = nil
   kNavCtlIsPreviewShowing  = 17,  // get preview status, parms<-Boolean
   kNavCtlAddControl  = 18,        // add one control, parms->ControlHandle
   kNavCtlAddControlList  = 19,    // add control list, parms->Handle (DITL)
   kNavCtlGetFirstControlID = 20,  // get 1st control ID, parms<-UInt16
   kNavCtlSelectCustomType  = 21,
                     // select custom menu item. parms->NavMenuItemSpec
   kNavCtlSelectAllType = 22,
                     // select an "All" menu item, parms->SInt16
   kNavCtlGetEditFileName = 23,
                     // get save dialog's file name, parms<-StringPtr
   kNavCtlSetEditFileName = 24
                     // set save dialog's file name, parms->StringPtr
};
typedef SInt32 NavCustomControlMessage;
```

Adding Your Controls

After the kNavCBCustomize message has been processed, your own interface elements can be added to any dialog after you receive the kNavCBStart message. The easiest way to set this up is to create a 'DITL' resource (in local coordinates relative to the customization area). Use the next example for this purpose:

```
gDitlList = GetResource ( 'DITL', 3000 );
theErr = NavCustomControl ( callBackParms->context,

kNavCtlAddControlList,gDitlList);
```

Then, when the dialog is being closed, respond to the kNavCBTerminate event message by disposing of this resource. Of course, a 'DITL' resource is not the only method. You can call NewControl and then send the ControlHandle to Navigation Services by using the next example. Likewise with a ControlHandle, be sure to dispose of it after receiving the kNavCBTerminate event message.

```
gCustomControl = NewControl( callBackParms->window,
                             &itemRect, "\pcheckbox", false,
                                  1, 0, 1, checkBoxProc, NULL );
theErr = NavCustomControl( callBackParms->context,
                             kNavCtlAddControl,gCustomControl);
```

Handling Your Controls

To work with your controls, use your event callback procedure to respond to mouseDown events. You can use the Dialog Manager to find out which one of your controls was clicked but you must use a designated offset to map the Dialog Manager's control ID to your custom controls. The next example shows how to respond to a mouseDown event:

```
void HandleCustomMouseDown( NavCBRecPtr callBackParms )
{
   OSErr          theErr = noErr;
   ControlHandle whichControl;
   Point          curMousePt;
   short          theItem = 0;
   UInt16         firstItem = 0;
   short          realItem = 0;
   SInt16         partResult = 0;

   GetMouse( &curMousePt );

   // find the control's item number:
   theItem = FindDialogItem( callBackParms->window,
                             curMousePt);
   // find the control itself:
   whichControl = FindControlUnderMouse(curMousePt,
                                 callBackParms->window, &partResult );
   if ( whichControl != NULL && partResult != 0 )
   {
      // get the item number of the control
      theErr = NavCustomControl( callBackParms->context,
                           kNavCtlGetFirstControlID, &firstItem );
      // map it to our DITL constants:
      realItem = theItem - firstItem + 1;

      // finally handle your control:
      switch ( realItem )
      {}
   }
}
```

Drawing Previews

Navigation Services gives you built in preview support for common preview types by using QuickTime's Preview components. These components are also used by the Image Compression Manager. If, however, you introduce new special file types to preview different types of data, you may want override the appearance of how file previews are displayed. To implement your own previews, set the kAllowPreviews flag in the browserOptions field of NavDialogOptions record. Then provide a preview callback routine to draw the preview. Your preview callback procedure gets called whenever an object is selected from the browser. I chose to do a simple preview example in **Figure 5** by drawing the file name and include an extra "Preview Info" button.

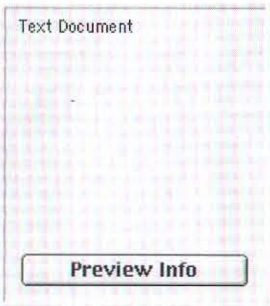


Figure 5. *A Customized Preview.*

Use the callBackParms->eventData->param field that contains an AppleEvent Descriptor of the file to preview. To use my preview info button, I respond to the kNavCBAdjustPreview event in my event callback procedure. This event determines if the preview area has been turned on or off (the user clicked Show/Hide Preview button), or if the preview areas has been resized. To check if preview is on or off, I use the next example:

```
Boolean previewShowing;
theErr = NavCustomControl( callBackParms->context,
                     kNavCtlIsPreviewShowing, &previewShowing );
```

To determine the preview area that was resized, I reference the callBackParms->previewRect field. This will require that I move my preview info button as a result of the resizing. The minimum preview area is 145 pixels wide by 118 pixels high.

Some file selections may require that I remove my preview info button. Using the callBackParms->eventData>param field, I can add or remove my button based, depending on if the selection is a file or a folder. Each time a different file object is clicked, I may want to have a preview with different sets of controls for each file type. So every time the preview callback procedure is called, I must decide to create and remove my individual controls accordingly.

In order to respond to my controls I have to call TrackControl in my event callback procedure. This is similar to the way you interact with your own controls in the customization area. Finally, you needn't worry about errors. Navigation Services will output any possible file system error on a given file to the upper left corner of the preview area.

```
pascal Boolean myPreviewProc( NavCBRecPtr callBackParms,
                              NavCallBackUserData callBackUD )
{
   OSErr          theErr = noErr;
   Boolean        previewShowing = false;
   AEDesc         resultDesc;
   FSSpec         previewFileSpec;
   Boolean        result = false;
   myGlobalsPtr   globals = (myGlobalsPtr)callBackUD;
   // find out of the preview area is showing:
   theErr = NavCustomControl( callBackParms->context,
                        kNavCtlIsPreviewShowing, &previewShowing);
   if ( theErr == noErr && previewShowing )
   {
      // make sure we have the right descriptor type:
      if ((theErr = AECoerceDesc(
                        (AEDesc*)callBackParms->eventData.param,
                        typeFSS, &resultDesc )) == noErr)
      {
         FInfo info;
         // get the FSSpec from the descriptor:
         BlockMoveData( *resultDesc.dataHandle,
                        &previewFileSpec, sizeof( FSSpec ) );
         if ((theErr = FSpGetFInfo(
                        &previewFileSpec, &info )) == noErr)
         {
            // the finder info was successfully retrieved, so we have a file to preview:
            short     saveTxFont;
            short     saveTxSize;
            GrafPtr   savePort;
            Rect      previewInfoButtonRect;
            // setup and create our Preview Info button:
            SetRect( &previewInfoButtonRect,
                     callBackParms->previewRect.left+10,
                     callBackParms->previewRect.bottom-30,
                     callBackParms->previewRect.right-10,
                     callBackParms->previewRect.bottom-10 );
            if ( globals->gPreviewInfoButton == NULL )
               // the control does not exist, so create one:
               globals->gPreviewInfoButton =
               NewControl( callBackParms->window,
                        &previewInfoButtonRect,
                          "\pPreview Info", false, 1, 0, 1,
                          pushButProc, NULL);
            saveTxFont = callBackParms->window->txFont;
            saveTxSize = callBackParms->window->txSize;
            GetPort( &savePort );
            SetPort( callBackParms->window );
            // draw our preview information:
            TextSize( 10 );
            TextFont( applFont );
            MoveTo(  callBackParms->previewRect.left + 8,
                     callBackParms->previewRect.top + 15 );
            DrawString( previewFileSpec.name );
            SetPort( savePort );
            TextFont( saveTxFont );
            TextSize( saveTxSize );
            // show our Preview Info button:
            if (globals->gPreviewInfoButton != NULL )
            {
               ShowControl( globals->gPreviewInfoButton );
               Draw1Control( globals->gPreviewInfoButton );
            }
            // return true since we handled drawing the preview:
            result = true;
         }
```

Power Too Abundant to Meter!

neo•logic®
Powering Development of Object-Oriented Applications

Powerful — NeoAccess is the most powerful object-oriented database engine available. It displays electrifying performance—up to ten times that of its competitors. Behind its elegant programming interface is a fully optimized relational query engine tuned for short access times and an object cache with automatic garbage collection for nearly instantaneous access to previously used objects. All in a memory footprint as small as 150K.

No Runtime Fees — Get the power of NeoAccess, and avoid the expense and administrative hassle of feeding the runtime fees meter. You pay one affordable price no matter how many copies of your application you sell or use.

Cross Platform — Others may promise cross-platform development tools—NeoLogic delivers. NeoAccess is a set of C++ classes designed for use with popular compilers and application frameworks on Windows®, Macintosh®, and Unix™ platforms. Source code is included so it can even be used with custom frameworks.

Proven — Thousands of developers, including America Online® and Claris®, have already found that NeoAccess enabled them to build fast, powerful internet applications in record time. That's why there are more copies of NeoAccess based applications on end-user machines than any other object database backend. Tap the power!

Save $100

neoAccess®
Cross-Platform Object Database Engine

Order Now!

M–F 9AM to 5PM Pacific:	1-800-919-6353
For Information and Customer Service:	1-510-524-5897

NeoLogic Systems, Inc.
1450 Fourth St., Suite 12, Berkeley, CA 94710
v. 510.524.5897 f. 510.524.4501 neologic@neologic.com

Order Directly from Our Web Site! www.neologic.com
Download the Architectural Overview

```
      else
      {
      // failed trying to get the file to preview, so remove our Preview Info button
      // since we don't plan to draw the preview:
         if ( globals->gPreviewInfoButton != NULL )
         {
            DisposeControl( globals->gPreviewInfoButton );
            globals->gPreviewInfoButton = NULL;
         }
      }
    }
  }
  return result;
}
```

Extending the Type Popup Menu

If you would like to add your own menu items to the type popup menu found in the Open and Save dialog boxes, use NavMenuItemSpec to describe each menu item you want to add. This lets you add specific document types to be opened, or different ways of saving a file (i.e. HTML format, Text with line breaks, etc.). The menu items can be added to the menu by adding a Handle to one or more NavMenuItemSpecs to the popupExtension field in the NavDialogOptions record.

To handle your custom popup menu items, respond to the kNavCBPopupMenuSelect event when Navigation Services calls your event callback procedure. It is up to you to interpret the menu item data and respond accordingly. Extending the popup menu also requires a filter callback procedure if you are to re-filter the browser after the menu selection is made.

Conclusion

As you can see, Navigation Services is a powerful and easy to use component in the Mac OS. At last you have a modern interface to navigate and organize your file system. Your experience with this technology will be rewarding from the start. It is easy to get started and once you have a basic familiarity with the API, you can then tackle the more advanced features. You will soon discover that writing a Navigation Services "savvy" application will be well worth the time and investment.

References

- "Programming with Navigation Services", (Apple Computer, 1998).
- Inside Macintosh, Volume 1, The Standard File Package, (Addison-Wesley Publishing, 1985).
- Inside Macintosh: Files, Standard File Package, (Addison-Wesley Publishing, 1992).
- Inside Macintosh: QuickTime Components, (Addison-Wesley Publishing, 1993).

Acknowledgements

Thanks to Sean Findley, Yan Arrouye, and Otto Schlosser.

Mac OS, Macintosh, QuickTime are registered trademarks of Apple Computer, Inc. PowerPC is a trademark of International Business Machines Corporation, used under license therefrom. Screenshots of NavGetFile and NavPutFile, © 1997-98, Apple Computer, Inc. All Rights Reserved.

MT

by Tom Djajadiningrat and Maarten Gribnau, Delft University of Technology
Edited by the MacTech Editorial Staff

Desktop VR using QuickDraw 3D, Part II

Using the Pointing device Manager Implementation of a HeadTracked Display

Summary

Wouldn't it be cool to be able to look around three dimensional objects displayed on your monitor by moving your head, just as if the objects were standing there? Kind of like a hologram, but with the flexibility of 3D computer graphics. Futuristic and expensive? It could be easier and cheaper than you think. In a two part article we explain how to implement such a system, also known as a head-tracked display, on a PowerMacintosh. It provides the user with a sense of depth without the use of stereoscopy. To facilitate implementation we use QuickDraw 3D, Apple's 3D graphics library. In terms of hardware all you need is a PowerMac with QuickDraw 3D, an absolute position-measuring device with three degrees of freedom and, preferably, a QuickDraw 3D accelerator board. This month we discuss the hardware related aspects of the head-tracked display. The graphics related topics were covered in last month's issue.

Introduction

This article is the second and last part of our coverage of a head-tracked display system. The system has two parts: the viewer application, called MacVRoom and a driver. In last month's graphics issue we focused on the viewer application and explained how to control the camera by the user's head position and to show the corresponding perspective on the monitor. This month, we will explain how to write the driver and how to use the Pointing Device Manager to handle the communication between the driver and MacVRoom. We will also discuss a number of calibration methods that will achieve the maximum accuracy. Illustrated in **Figure 1** are the topics covered in both articles. In last month's issue we suggested which topics to read or skip depending on your goals. We also addressed the issue of which knowledge and hardware you require.

Tom Djajadiningrat (J.P. Djajadiningrat@io.TUDelft.nl) is an industrial designer interested in products and computers which are intuitive in use. When not trying to convince others that he will now finish his PhD thesis on interfaces using head-tracked displays 'really soon', or hassling Maarten and the QuickDraw 3D mailing list with silly programming questions, he dreams of designing the 25th anniversary Macintosh.

Maarten Gribnau (M.W. Gribnau@io.TUDelft.nl) is an electrical engineer interested in Computer Graphics and Interaction Design. He has successfully delayed Tom's research project so they can now both convince others that they will complete their PhD theses 'really soon'. Apart from his research on two-handed interfaces for 3D modeling applications, he occasionally drinks a strong cup of Java when he is trying to program the ultimate internet golf game.

Road Map to This Article

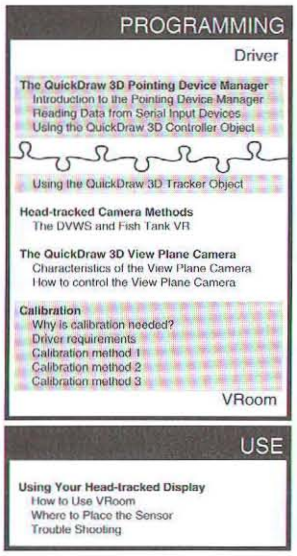

PROGRAMMING
Driver
The QuickDraw 3D Pointing Device Manager
Introduction to the Pointing Device Manager
Reading Data from Serial Input Devices
Using the QuickDraw 3D Controller Object
Using the QuickDraw 3D Tracker Object
Head-tracked Camera Methods
The DVWS and Fish Tank VR
The QuickDraw 3D View Plane Camera
Characteristics of the View Plane Camera
How to control the View Plane Camera
Calibration
Why is calibration needed?
Driver requirements
Calibration method 1
Calibration method 2
Calibration method 3
VRoom
USE
Using Your Head-tracked Display
How to Use VRoom
Where to Place the Sensor
Trouble Shooting

Figure 1. *Overview of the two-part article. The subjects which are covered this month are marked with a grey block.*

The QuickDraw 3D Pointing Device Manager

Introduction to the Pointing Device Manager

The MacVRoom application needs to read positions from a 3D input device to establish the position of the head of the user. We could have chosen to use one primary position sensing device and to include the code for reading the device into MacVRoom, but we would rather support any 3D input device. This is one of the reasons for using QuickDraw 3D's Pointing Device Manager (PDM). It enables you to separate the code for reading device positions from the application itself. Without the PDM driver code would have to be added to the application for each new device to be supported. Using the PDM, only the driver of the device needs to be written and every PDM-savvy application will be able to use it. The next few paragraphs will explain the basic concepts of the PDM.

The PDM was added to the QuickDraw 3D library to provide a means for 3D applications to support 3D pointing devices. Most computer systems come equipped with 2D pointing devices (like a mouse, a trackball or a graphics tablet) that measure positions in two dimensions. 3D pointing devices come in a lot of flavors but they are generally able to sense more than two independent variables. Most operating systems do not provide a way to transport data from 3D pointing devices to 3D applications. The PDM does this, and in addition, it supplies routines that you can use to determine what kind of devices are available and to assign them to tasks in your application.

The PDM defines two types of objects: controllers and trackers. Controller objects are the PDM's abstract representation of 3D pointing devices. They are intended to be created by device drivers. Trackers are created by a 3D application to track positions, orientations and button states of a pointing device. The purpose of this is that controllers and trackers can be in different applications. A driver creates a controller and updates it regularly with fresh device readings, while applications create a tracker, connect it to the controller and extract the device readings from it when needed.

Figure 2 shows the information streams that flow between connected controllers and trackers. The main objective of the controller object is that pointing devices can send their position and orientation and the state of their buttons to the tracker. The figure also shows two other data streams. The controller values are meant to report about optional device dependent information to the application, like the state of switches. Controller channels can be used to communicate data from the application back to the device. Some devices for instance, are equipped with alphanumeric displays that label the switches. The channels can be used to have the driver update these labels. Although controller channels and values may be useful in some applications, their use has the disadvantage that the application will have to contain device dependent code. This eliminates device independence of the application which is a motivation for using the PDM.

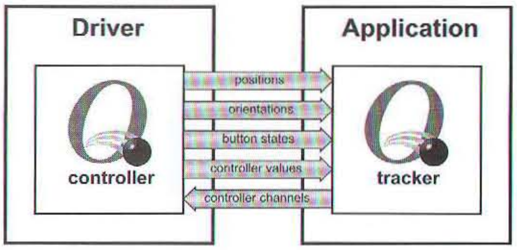


Figure 2. *Information streams between QuickDraw 3D Controller and Tracker objects.*

Positions and orientations can be reported to the tracker in two ways, absolute and relative. This is because there are two kinds of pointing devices. Absolute pointing devices measure positions and/or orientations, relative to the origin of their coordinate system. Relative pointing devices do not have an origin, they report only the changes in position and/or orientation since the last measurement. This is best illustrated by comparing two 2D input devices: a drawing tablet and a mouse. The tablet is an absolute device. When you pick up the stylus and move it to another location, the cursor will move to another location too because the tablet reports the new location to the system. A mouse however, does not report the new location to the system when it is picked up and moved to a new location. While the mouse is in the air the tracking is lost and it can not report changes in position to the system until it is put down again. In this project we need to establish the position of the head of the user relative to the monitor and therefore we need an absolute pointing device.

Using the PDM, you can use several input devices concurrently to control different application tasks. A useful application area of this feature is two-handed input where the simultaneous input of two hands is used (Gribnau and Hennessey, 1998). The PDM also enables you to connect multiple controllers to the same tracker. This means that several input devices can control one application task at the same time.

The reverse situation is prohibited; it is not allowed to connect one controller to more than one tracker. In practice, connecting several devices to one task is useful only when, at most, one of those is absolute. When two or more absolute controllers are connected to one tracker they will compete for the current position and/or orientation of the tracker resulting in a jittery appearance. Relative devices do not behave dominantly. The connection of a few relative devices to one tracker will give predictable results. The changes in positions and/or orientations they report to the tracker are added to the current position and/or orientation of the tracker. The behavior of the combination of an absolute device and one or more relative devices is logical too. The offsets reported by the relative device(s) are overruled by the measurements of the absolute devices.

To illustrate the use of the PDM we will now describe how to create a QuickDraw 3D driver for a serial device and how to access it from the MacVRoom application. In the code presented we have removed all the device related parts and all the code concerning the user interface. Readers interested in the omitted code can download the source code of the drivers that we offer on the web site. The first section will deal with the reading of positions from a device. The second section will explain how to use the PDM controller object to report the positions to an application and in the last section you will learn how to use the QuickDraw 3D tracker object in an application to read positions from a QuickDraw 3D device driver.

Reading data from serial input devices

The main objective of the driver is to execute as many readings from the serial port as needed for a smooth operation. As the driver is running in the background we cannot poll the serial port periodically for new data because we are not sure we will get enough processor cycles. Instead of polling we will therefore use the Device Manager PBReadAsync routine to read the serial port. This routine can be used to request a number of bytes from the serial port and it will call a completion routine when they have arrived in the buffer. Here, we would like to extract the positions from the bytes in the buffer but regrettably we can't. Calling routines that move memory will crash the driver and it is not advisable to do lengthy operations that could block other operations. Hence, we have to find a way to process the data at a later time when it is safe to call lengthy routines that move memory around. This is exactly the functionality the Deferred Time Manager offers. You can use its services whenever you need to install a lengthy interrupt task capable of running with all interrupts enabled. In the completion routine we install a deferred task that will process the data and update the controller. At the end of the deferred task we request another reading from the serial port. This closes the circle of a read chain that will read data from the serial port and process it indefinitely. The read chain is illustrated in **Figure 3** and demonstrates the sequence of events.

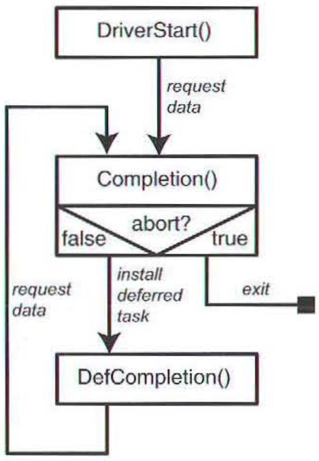


Figure 3. *The driver serial port read chain.*

Listing 1 contains constants, global variables and the code for the DriverStart routine. The routine prepares the global variables and the structures for both the completion routine (fIoParam) and the deferred task (fDefTask). The pointer to our completion routine is put in the ioCompletion field of the IOParam structure. The rest of the fields tell the Device Manager how much data is requested, from which device and were it should be stored. You may notice that the symbol PACK_LEN is not defined in the code. It is the number of bytes that the device needs to send a new reading and therefore different for each device.

To set up the DeferredTask structure we need to fill in only the qType, dtAddr, and dtReserved fields. The dtAddr field specifies the address of the routine whose execution you want to defer, we set it to our deferred completion routine. The dtParam field contains an optional parameter that is loaded into register A1 just before the routine specified by the dtAddr field is executed, we don't use it and clear it. The dtFlags and dtReserved fields of the deferred task record are reserved and the last one should be set to 0. At the end of the routine the serial port buffer is cleared, using SerRecvFlush (not listed), and a request is made for the first package of bytes.

Listing 1: Driver.c

DriverStart

```
#define BUF_LEN       256

//The state of the driver
static Boolean          fRunning = false;
//The serial input port ID
static short            fSerInp = 0;
//The buffer that holds the bytes read from the port
static Byte           fBuf[BUF_LEN];
//The number of valid data bytes in the buffer
static long           fBufCount = 0;
//The completion routine device parameter block
static IOParam          fIoParam;
//The deferred task manager structure for our deferred completion routine
static DeferredTask     fDefTask;
//The current position of the device
TQ3Point3D              fPos;

OSErr DriverStart()
{
  OSErr err;
  // Set up the serial parameter block
  fIoParam.ioCompletion  = NewIOCompletionProc(Completion);
  fIoParam.ioRefNum      = fSerInp;
  fIoParam.ioPosMode     = fsAtMark;
  fIoParam.ioPosOffset = 0;
  fIoParam.ioBuffer      = (Ptr) fBuf;
  fIoParam.ioReqCount    = PACK_LEN;

  // Set up the deferred task block
  fDefTask.qType         = dtQType;
  fDefTask.dtAddr        = NewDeferredTaskProc(DefCompletion);
  fDefTask.dtParam   = 0;
  fDefTask.dtReserved  = 0;

  // Flush the serial input port
  err = SerRecvFlush(fSerInp);
  if (err != noErr) goto exit;
  fBufCount = 0;

  //Ask for the first package
  err = PBReadAsync((ParmBlkPtr) &fSerParm);
  if (err != noErr) goto exit;
  fRunning = true;

exit:
  return err;
}
```

Listing 2 contains the Completion routine called when the Device Manager has read the number of bytes requested or when it has encountered an error. The error is passed through the ioResult field of the IOParam structure. If it is equal to abortErr there has been a request to cancel the io operations involving the serial port. In this case we will have to end the read chain by not installing the deferred completion task. On all other errors, we clear the buffer by setting the buffer size fBufCount to zero. It there was no error, fBufCount is updated. At the end of the routine the deferred completion task is installed using the DTInstall call.

Listing 2: Driver.c

Completion

```
static pascal void Completion(ParmBlkPtr /*paramBlock*/)
{
  if (fIOParm.ioResult == abortErr) {
    // KillIO was called, end the read chain
    return;
  }
  // Update the buffer byte count
  fBufCount += fIOParm.ioActCount;
  if (fIOParm.ioResult != noErr) {
    // Reset the buffer byte count on serial error
    fBufCount = 0;
  }
  // Install the deferred task to process the bytes in the buffer
  DTInstall(&fDefTask);
}
```

In Listing 3, you will find the deferred completion routine. This is the routine that actually processes the bytes read from the device and issues another read. The first thing the routine does is to save the current position of the device. Then it calls the DecodeBuffer routine (not listed) that is specific to the device the driver is created for. It is expected to decode the current position from the bytes in the buffer and to store it in the global variable fPos. During the decode the routine might notice that it is out of sync and needs more bytes to decode a package. The routine should then shift the remaining data bytes to the start of the buffer and reports their number by updating fBufferCount. After decoding the bytes, the controller is updated by calling UpdateController (discussed later), but only if the current position has changed.

The end of the routine establishes the number of bytes to read next and issues a request for them to the Device Manager. Normally the number of bytes requested will be equal to the number of bytes that the device uses to encode a position and/or orientation (defined by the symbol PACK_LEN). If there are bytes left in the buffer, we subtract their number from the number of bytes requested. This should get the decoding algorithm in sync with the device data. The IOParam structure is updated with the requested number of bytes and a pointer to the right position in the data buffer. In case the buffer is overrun, the buffer is reset by setting fBufCount to zero and a request for a new package is made to get in sync again.

Listing 3: Driver.c

DefCompletion

```
static pascal void DefCompletion(long /* dtParam */)
{
  //The number of bytes requested next
  long req;
  // Save the current position
  TQ3Point3D oldPos = fPos;

  // Decode the coordinates in the buffer to fPos, update the number of bytes
  DecodeBuffer(fBuf, &fBufCount);
  if ((fPos.x != oldPos.x) ||
      (fPos.y != oldPos.y) ||
      (fPos.z != oldPos.z)) {
    //The position has changed, update the controller
    UpdateController();
  }
  // Calculate the number of bytes that will be requested next
  req = PACK_LEN - fBufCount;
  if ((req > 0) && (fBufCount < (BUF_LEN - req))) {
    //The number of bytes requested fits in the buffer
    fIOParm.ioBuffer   = ((Ptr) fBuf) + fBufCount;
    fIOParm.ioReqCount = req;
  }
  else {
    // Error situation, reset the buffer
    fIOParm.ioBuffer   = (Ptr) fBuf;
    fIOParm.ioReqCount = PACK_LEN;
    fBufCount = 0;
  }
  // Request the next few bytes
  PBReadAsync((ParmBlkPtr) &fSerParm);
}
```

Using the QuickDraw 3D Controller Object

At this point the current position of the device is established. The next step is to create the controller object that can be polled by trackers. Listing 4 illustrates how to create the controller. The routine sets the fields of the PDM controller data structure and uses the Q3Controller_New routine to have QuickDraw 3D create the controller object. The signature field of the controller structure is a C-type string that applications can use to distinguish between controllers. As we choose not to support any controller values or channels the other fields are set to nil.

Listing 4: Driver.c

CreateController

```
static TQ3ControllerObject CreateController()
{
  TQ3ControllerData controllerData;

  controllerData.signature        = "Acme Driver";
  controllerData.valueCount       = 0;
  controllerData.channelCount     = 0;
  controllerData.channelGetMethod = NULL ;
  controllerData.channelSetMethod = NULL ;
  return Q3Controller_New(&controllerData);
}
```

The controller object is subsequently used by the UpdateController routine in Listing 5. The routine checks whether the controller is currently affecting the 2D system cursor. As you remember from the introduction to the PDM, trackers can be connected to controllers. If a controller is not connected to a tracker, the PDM will connect it to the 2D system cursor. This is nice for relative pointing devices, because you can use your 3D pointing device to operate your Mac. For absolute devices however, this is not so convenient. Depending on the coordinates the driver generates, it may park the cursor in a corner of the screen and make your Mac inoperable. The driver therefore uses Q3Controller_Track2DCursor to check whether the controller is connected to the system cursor. The PDM sets track2DCursor to kQ3False when the controller is not connected to the cursor, only then the current position is entered into the controller object. Devices that measure orientations and/or button states can pass their data in the same way.

Listing 5: Driver.c

UpdateController

```
static void UpdateController()
{
  TQ3Boolean track2DCursor;
  (* Check whether the controller is connected to the system cursor *)
  Q3Controller_Track2DCursor(fController, &track2DCursor);
  if (track2DCursor == kQ3False) {
    Q3Controller_SetTrackerPosition(fController, &fPos);
  }
}
```

Using the QuickDraw 3D Tracker Object

Our MacVRoom application needs to read the positions from a 3D pointing device to update the camera. It accomplishes this by creating a PDM tracker that can connect to controllers on the system. The question is, how does it find a controller and to which controller does it connect? The PDM provides a way to search for all the controllers present. Using this functionality the application's tracker can be connected to all the controllers active on the system. When connecting multiple controllers to one tracker, you might encounter the strange behavior that was discussed in the introduction to the PDM. Listing 6 presents the CreateTracker routine that implements this functionality.

First it creates a tracker. You may wonder why it passes NULL to the Q3Tracker_New routine. This is because there are two ways to find out if the position of a tracker has changed. If you provide a pointer to a notification routine in the Q3Tracker_New call, the routine will be executed by the PDM every time the position changes. The notification threshold can be changed, so you can set the distance the device will have to move before the notification routine is called. Our application does not use a notification routine and notifies the PDM of this fact by passing NULL. It polls the tracker instead to see whether the position of the tracker has changed.

Once the tracker is created, the CreateTracker routine connects it to the controllers it finds. The PDM offers the Q3Controller_Next call to loop through the controllers. The first parameter must be set to NULL to find the first controller. A reference to the controller is returned in the second parameter. This can be fed again into the first parameter to find the next controller. If the PDM can not find a controller the reference parameter returned is set to NULL. This is the sign that the end of the list is reached. If a controller is found, then it is connected to the tracker and the boolean variable foundController is updated. The variable is used at the end of the routine to check whether there were controllers present on the system. If no controllers were found, the routine disposes of the tracker and returns NULL. This way, MacVRoom knows there is no controller available and it will enable mouse control. If a controller was found, MacVRoom can use the tracker to extract the device coordinates. This will be discussed later when the adjustment of the camera parameters is covered.

Listing 6: (De)Init.c

CreateTracker()

```
TQ3TrackerObject CreateTracker()
{
  TQ3TrackerObjecttracker;
  TQ3Status        status;
  TQ3ControllerRefref;
  TQ3Boolean          foundController = kQ3False;

  tracker = Q3Tracker_New(NULL);
  status = Q3Controller_Next(NULL, &ref);
  while ((ref != NULL) && (status == kQ3Success)) {
    Q3Controller_SetTracker(ref, tracker);
    foundController = kQ3True;
    status = Q3Controller_Next(ref, &ref);
  }
  if (foundController == kQ3False) {
    Q3Object_Dispose(tracker);
    tracker = NULL;
  }
  return tracker;
}
```

Concluding our coverage of the PDM a warning is issued to readers that are planning to use our code in their own applications. In theory the PDM allows you to dynamically assign controllers to trackers. In practice however, you will probably encounter a bug when you try to connect a tracker to a controller a few times. If you are running with the QuickDraw 3D GM extensions, your application will crash. At the moment the only solution is to connect trackers to controllers once. Hopefully the QuickDraw 3D team will solve this problem in their next release.

Calibration

Why is calibration needed?

Calibration would be simple if the sensor could be put in the middle of the eye, the origin of the tracker base unit could be put in the middle of the pane, and the driver would report the position of the sensor in inches. All that would be required to find the position of the camera in world coordinates, would be to divide the sensor position by the width of the rendering pane in inches. Unfortunately, things are not that simple as the sensor cannot be mounted in the eye, and the tracker cannot be placed in the monitor.

For a movement parallax system based on the fishtank projection method calibration is quite important. If we get it wrong the virtual camera position will not correspond to the user's head position and as a result he will see a distorted image. One of the problems is that we do not know the dot pitch of your monitor and the current resolution. Even though we know the size of the rendering pane in pixels we do not know what its size will be in inches on your monitor.

Driver requirements

To simplify matters we will make three assumptions about our driver (the drivers which accompany this article conform to these assumptions. If you are writing a driver yourself, please make sure that it does so too). The first one is that it results in a right handed coordinate system with the Y-axis pointing upwards and the Z-axis pointing towards the user (see for example Figure 6). The second one is that the tracker is isotropic in X, Y and Z or that is has been made to appear isotropic to MacVRoom by appropriate scaling in the driver. For example, if the sensor is moved over a certain distance along the X-axis, the change in X-coordinate should be the same as the change in Y-coordinate when the sensor is moved over the same distance along the Y-axis. The third assumption is that the driver has scaled the tracker coordinates to inches. This means that if we move the sensor one inch along one of the axes the coordinate of that axis will change by one.

There are a number of methods to do the calibration. We will discuss three methods in order of increasing complexity and accuracy. In the first method, the set-up is measured with a ruler and calibration is performed once only at compilation time. In the second method, the user is required to hold the sensor in two locations, expressed in terms of the pane width, when the app is launched. In the third method, the user is required to hold the sensor in three locations of which the exact position relative to the pane is known. The three methods are summarized in **Table 1**. You may wish to refer to the decision chart (**Figure 4**) from time to time while you read the description of the three calibration methods. No matter what calibration method you choose, you should, of course, not burden the user with calibration every time MacVRoom is run. Although MacVRoom can neither save the calibration settings and nor recalibrate without quitting first, a real application should offer this functionality.

INCREASING COMPLEXITY ▼	*source code supplied*	*measurements*	*tracker & pane coord. sys. must share orientation?*	*accuracy*	*disadvantages*
method 1	no	• O_tC_p • pane width	no	high	• tracker origin required
method 2	yes	none	no	• low without cardboard template • high with cardboard template	• low accuracy w/o template • high accuracy requires cardboard template
method 3	yes	P_0C_p	yes	high	• cardboard template

Table 1. *Characteristics of the three calibration methods.*

GOT BUGS?

...and you're drowning in paper trying to keep track of them?

You need **BugLink!**

- **BugLink is a client/server application** allowing you to connect developers, testers, and support engineers anywhere in the world.
- **Intuitive user interface** gives you the information you need at a glance.
- **Fully customizable database**—add the fields you need to each project.
- **Custom TCP/IP protocol minimizes network traffic**—ideal for dial-up connections.
- **Client applications can operate 'off-line'**—allowing bug entry and modification even when not connected to the network!
- **Cross platform**—set up mixed Macintosh and Windows environments in minutes with no additional software needed!
- **Try before you buy.** Download and try risk free for 30 days from http://www.pandawave.com/bl/

A 5 User License starts at $299; that's only $60 per person.

The Panda*Wave*

http://www.pandawave.com

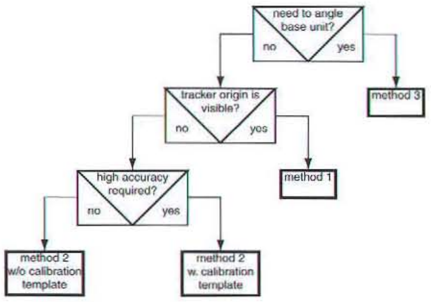

need to angle base unit?
no
yes
method 3
tracker origin is visible?
no
yes
method 1
high accuracy required?
no
yes
method 2 w/o calibration template
method 2 w. calibration template

Figure 4. *Decision flow chart for the three calibration methods.*

Calibration Method 1

In the first method we measure the distance in inches from the tracker origin to the pane center (OtCp) along the coordinate axes. We also measure the width of the pane in inches. From the driver we get the position of the sensor relative to the tracker origin in inches (OtP). The position of the sensor relative to the pane center (CpP) can now be found by CpP = OtP-OtCp (**Figure 5**). We have made life easier by making the image which the View Plane Camera takes out of the view plane one QuickDraw 3D unit wide. Since this image takes up the full width of the rendering pane, and the driver provides measurements in inches, we can find the position of the virtual camera in virtual world coordinates by scaling the position of the sensor relative to the pane center by dividing it by the pane width in inches. Thus the top-left corner of the pane in virtual world coordinates is {-0.5, 0.71, 0} and the bottom-right corner is {0.5, -0.71, 0}.

This calibration method is the easiest but has two disadvantages. The first is that the tracker origin needs to be known. With some trackers, for example the DynaSight, this is indeed the case as the origin is indicated on the base unit. However, with others, for example the FreeD, we do not know where the origin is located. The second disadvantage is that the tracker coordinate system needs to be aligned to the pane coordinate system. The base unit of the tracker thus needs to be accurately aligned with the monitor screen. Though theoretically it is possible to take the orientation of the tracker into account, in practice it is difficult to accurately measure its orientation with a protractor. For these reasons we have not implemented this method.

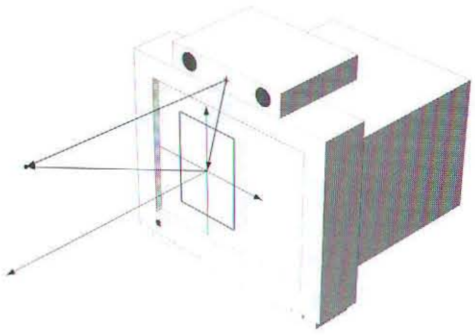


Figure 5. *In calibration method 1 the vector from the center of the pane (Cp) to the sensor (P) is found by CpP = OtP-OtCp, in which Ot is the origin of the tracker coordinate system. Note that the center of the pane need not coincide with the center of the screen.*

Calibration Method 2

The second method is the one which MacVRoom uses by default. The user is asked to keep the sensor twice the width of the pane in front of the top-left corner of the pane (**Figure 6**), press return and repeat the process for the bottom-right corner. Listing 7 shows how from these two 3D points the transformation matrix can be calculated. First we find the pane center relative to the tracker origin and the pane width in tracker coordinates. From these we can find the translation and scaling matrices, which are then combined into a composite calibration matrix. Note how we're not bothered anymore about the location of the tracker origin. Therefore you can simply hook up a different tracker, restart MacVRoom and calibrate the system for the new tracker. You can also switch screen resolution and work with a different size pane as this method takes pane width into account. This method also has some disadvantages. As with the first method the tracker and pane coordinate system need to have the same orientation. A second disadvantage is that it is difficult to keep the sensor perfectly in front of the window by twice the pane width, though this can be overcome with the help of a cardboard template. Still, we are providing you with this method because it gets you up and running quickly as - in its basic form - it does not require you to build any cardboard templates.

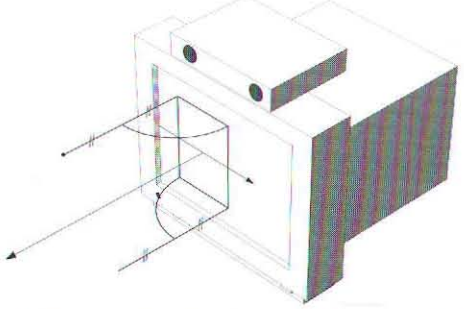


Figure 6. *The first calibration point (P1) is twice the pane width in front of the top left corner of the pane. The second calibration point (P2) is twice the pane width in front of the bottom right corner of the pane.*

Listing 7: Calibration.c

CalibrateTracker2
```
// From the two 3D points we can get the necessary info to do the calibration.

// Average the Z-coordinates of the two calibration points
calPoint1.z = calPoint2.z = (calPoint1.z+calPoint2.z)/2;

// Calculate width and height of window in raw tracker units
calWidth = calPoint2.x - calPoint1.x ;
calHeight = calPoint2.y - calPoint1.y ;

// Find matrix for sensor to eye center offset
Q3Matrix4x4_SetTranslate(&transSensorMatrix, 0, -kDy, 0);

// Calculate matrix to translate pane center to tracker origin
Q3Matrix4x4_SetTranslate(&transCpOtMatrix,
                            -(calPoint1.x + calWidth/2.0),
                            -(calPoint1.y + calHeight/2.0),
                            -calPoint1.z + 2*calWidth);

// Calculate scale matrix to scale to width = 1
Q3Matrix4x4_SetScale(  &scaleMatrix,
                       1/calWidth,
                       1/calWidth,
                       1/calWidth);

// Calculate the calibration matrix:
// Translate -OtCp, translate for sensor offset and scale.
Q3Matrix4x4_Multiply(  &transSensorMatrix,
                       &transCpOtMatrix,
                       &theDocument->fCalMatrix);

Q3Matrix4x4_Multiply(  &theDocument->fCalMatrix,
                       &scaleMatrix,
                       &theDocument->fCalMatrix);
```

Calibration Method 3

The main advantage of the third calibration method is that the orientation of the tracker coordinate system and the screen coordinate system need not be the same. This means that you have the freedom to place the tracker so that it looses track of the user as little as possible. For example, the volume wherein the Dynasight base unit can track its sensor is shaped like a cone, which emanates from the tracker base unit. To make sure the user stays within this cone, you may wish to reposition and reorient the tracker base unit. For example, in **Figure 7** the tracker base unit has been put under an angle.

To activate the third method change the conditional compiler statement "#define CALIBRATIONMETHOD3 0" to "#define CALIBRATIONMETHOD3 1" in MyDefines.h. This method does require some handiwork. You need to make a cardboard template cardboard template on which to put the monitor. On the cardboard template mark out three points (P0, P1, P2) forming a right angle as shown in **Figure 7**. P0P1 must be parallel to the screen coordinate system's X-axis, and the P0P2 must be parallel to the screen coordinate system's Y-axis. To specify your set-up you will need to enter the vector from P0 to the center of the pane (P0Cp) and the width of the pane in inches in CalibrateTracker3. On start up you need to hold the tracker in the three positions and press return after each one. Now we have these three points in the tracker coordinate system. From the three points we can calculate unit vectors in the x and z directions of the screen coordinate system (Listing 8). A unit vector in the Y direction of the screen coordinate system is found by taking the cross-product of the unit vectors in the x and z direction. We can find the translation vector from the tracker origin to the center of the pane (OtCp) by adding the vector P0Cp expressed in the tracker coordinate system to P0. The orthonormal vectors (i.e. unit vectors which are perpendicular to each other) expressed in the tracker coordinate system need to rotate into the unit axes of the screen coordinate system. They therefore form the first three columns in the rotation matrix (If you are reading up on this, remember that many books on computer graphics, for example the highly recommended Foley et al. (1995), use post-multiplication by column vectors, while QuickDraw 3D uses pre-multiplication by row vectors. To go from one convention to the other you need to transpose the matrices and reverse their order: [AB]t = BtAt). Scaling can be done by dividing by the pane width in inches, which will give us a virtual world coordinate system in 'pane width units'.

As long as the pane is not moved and the monitor is not moved with respect to the cardboard template, there is no need to change the statements defining the vector P0Cp and the width of the pane. Re-enter the three calibration points and the set-up is calibrated again.

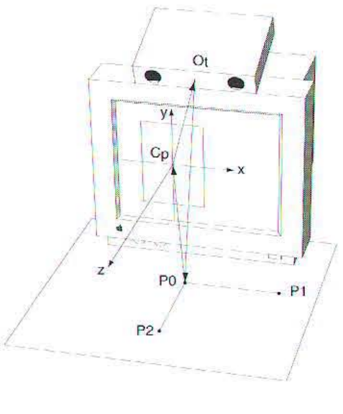


Figure 7. *For calibration method 3 a cardboard template with three calibration points (P0, P1, P2) is put underneath the monitor.*

NetProLive – "The Web Site"

Like the magazine, NetProLive the web site, will feature content for web developers and network administrators. Although selected material from NetProfessional magazine will appear on the site, it won't be just a static set of archives, but rather a dynamic set of pages.

NetProLive also contains industry news links to support resources such as mailing lists, community resources like User Groups and Professional Associations for web developers and network administrators, and much more.

NET NP PROFESSIONAL™

http://www.netprolive.com

Listing 8: Calibration.c

CalibrateTracker3

```
//parallel to X
Q3Point3D_Subtract(&P1, &P0, &Xraw) ;

//X normalized
Q3Vector3D_Normalize(&Xraw, &Xnorm) ;

//parallel to Z
Q3Point3D_Subtract(&P2, &P0, &Zraw) ;

//Z normalized
Q3Vector3D_Normalize(&Zraw, &Znorm) ;

//find the y vector which is perpendicular to x and z
Q3Vector3D_Cross(&Znorm, &Xnorm, &Ynorm) ;

// find OtCp in the tracker coordinate system.
Q3Vector3D_Scale(&Xnorm, POCp.x, &tempVector);
Q3Point3D_Vector3D_Add(  &P0,
                         &tempVector,
                         &tempPoint) ;

Q3Vector3D_Scale(&Ynorm, POCp.y, &tempVector);
Q3Point3D_Vector3D_Add(  &tempPoint,
                         &tempVector,
                         &tempPoint) ;

Q3Vector3D_Scale(&Znorm, POCp.z, &tempVector) ;
Q3Point3D_Vector3D_Add(  &tempPoint,
                         &tempVector,
                         &OtCp) ;

// find matrix for sensor to eye center offset
Q3Matrix4x4_SetTranslate(&transSensorMatrix, 0, -kDy, 0) ;

// find matrix for center pane to tracker origin vector
Q3Matrix4x4_SetTranslate(&transOtCpMatrix,
                              -OtCp.x, -OtCp.y, -OtCp.z) ;

// Set up the rotation matrix
// Make sure the rotation matrix is set to the identity matrix
// before we start modifying it.
Q3Matrix4x4_SetIdentity(&rotMatrix) ;

// each column vector of the rotation matrix
// is rotated by the rotation matrix to lie
// on positive x, y and z axes.
rotMatrix.value[0][0] = Xnorm.x;
rotMatrix.value[0][1] = Ynorm.x;
rotMatrix.value[0][2] = Znorm.x;
rotMatrix.value[1][0] = Xnorm.y;
rotMatrix.value[1][1] = Ynorm.y;
rotMatrix.value[1][2] = Znorm.y;
rotMatrix.value[2][0] = Xnorm.z;
rotMatrix.value[2][1] = Ynorm.z;
rotMatrix.value[2][2] = Znorm.z;

// find scale matrix to scale inches to 'pane width units'
scale = 1/paneWidthInches;
Q3Matrix4x4_SetScale(&scaleMatrix, scale, scale, scale);

// translate -OtCp, rotate, translate for sensor offset, and finally scale
Q3Matrix4x4_Multiply(  &transOtCpMatrix,
                       &rotMatrix,
                       &theDocument->fCalMatrix);

Q3Matrix4x4_Multiply(  &theDocument->fCalMatrix,
                       &transSensorMatrix,
                       &theDocument->fCalMatrix);

Q3Matrix4x4_Multiply(&theDocument->fCalMatrix,
                       &scaleMatrix,
                       &theDocument->fCalMatrix);
```

Conclusions

In this month's Part II we covered the hardware related aspects of a head-tracked display on PowerMacintosh using QuickDraw 3D. We showed you how to use the Pointing Device Manager to handle communication between a viewer application and a three degrees of freedom position tracker. The Pointing Device Manager has the potential to enrich 3D interaction through input devices other than keyboard, mouse or trackball. Also, we covered some calibration methods that can be used for accurate operation of the head-tracked display system. In the two articles we hope to have shown you how the QuickDraw 3D API greatly facilitates the implementation of virtual reality systems "for the rest of us". What are you waiting for?

Acknowledgements

We gratefully acknowledge P.J. Stappers, J.M. Hennessey, C.J. Overbeeke and A. van der Helm, for their constructive criticism during the preparation of this article.

Bibliography and References

- Apple Computer, Inc. (1992). Inside Macintosh: Processes. Reading, MA: Addison-Wesley Publishing Company.
- Apple Computer, Inc. (1994). Inside Macintosh: Devices. Reading, MA: Addison-Wesley Publishing Company.
- Ascension Technology Corporation, P.O. Box 527, Burlington, VT 05402, USA, tel. (802)860-6440 or (800)321-6596 (inside the USA only), fax. (802)860-6439, email *<productsales@ascension-tech.com>,<http://www.ascension-tech.com>*.
- Foley, J.D., Van Dam, A., Feiner, S.K. and Hughes, J.F. (1996). Computer Graphics Principles and Practice, Second Edition in C. Reading, MA: Addison-Wesley Publishing Company.
- Gribnau, M.W. and Hennessey, J.M. (1998). Comparing one- and two-handed input in a 3D assembly task. Proceedings of CHI'98, accepted.
- Origin Instruments Corporation, 854 Greenview Drive, Grand Prairie, TX 75050, USA, tel. 972-606-8740, fax. 972-606-8741, email *<sales@orin.com>*, *<http://www.orin.com>*.
- Owl by Pegasus Technologies Ltd., Merkazim 2000, 5 Hazoref st., Holon 58856, Israel, tel.: +972-(3)-5500633, fax: +972-(3)-5500727, email *<pegasus@pegatech.com>*, *<http://www.pegatech.com>*.
- Quickdraw 3D home page,*<http://www.apple.com/quicktime/qd3d/*.

MT

FROM THE FACTORY FLOOR

by Howard Hinnant and Dave Mark, ©1998 by Metrowerks, Inc., all rights reserved.

The New C++ Standard: Locales

In last month's column, we took our first pass at the Final Draft International Standard for C++. Specifically, we covered the topic of namespaces. In this month's column, Howard Hinnant is back, and will take us through the concepts behind C++ locales.

Howard Hinnant is a software engineer on the MSL team at Metrowerks, and is responsible for the C++ and EC++ libraries. Howard is a refugee from the aerospace industry where FORTRAN still rules. He has extensive experience in scientific computing including C++ implementations of linear algebra, finite difference and finite element solvers.

Dave: What is a C++ locale and why would I want to use one?

Howard: A locale is a C++ class in the standard library that allows you to easily customize I/O so that users of your software feel at home, wherever that home may be. For example, let's say you're writing some high finance report software. This stuff is for executives, and you know how much they like their formatting. So when you print out ten thousand bucks, it has to look like:

```
$10,000.00
```

Ok, so you've put 9 months into this software, and your boss is really pleased with it. In fact, he/she likes it so much, they want to use it for the offices world wide. Except that the formatting needs to be customized for each country... (you did plan for this didn't you!). The French executives might want to see:

```
10.000,00 Franc
```

And the German executives might want to see:

```
10.000,00 Mark
```

Does this mean you need seperate formatting routines for every country? No. This is precisely what locale was meant to solve. In fact, with locale it is not difficult to have several formats current at the same time (think international client/server).

Dave: But wait a minute, US$10,000 is not equivalent to 10.000,00 German Marks!

Howard: Good point. An important thing to understand about locale is that it only helps you with formatting and I/O. Any calculations (like exchange rates) are totally up to you. This all might become more clear with a specific example. Let's say you wrote a class called Money in your killer app that looked something like:

```
class Money
{
public:
Money(double amount) : amount_(100*amount) {}
double to_double() const {return amount_;}
private:
double amount_;
};

std::ostream& operator<< (std::ostream& os,
                                      const Money& amount);
```

Your top level code might look like:

```
Money amount;  //Your Money class
is >> amount;  // input from stream (perhaps from US)
os << amount;  // output to stream (perhaps to France)
```

Dave: How do these streams know which country to format for? Where is the formatting logic?

Howard: You (the programmer) do have to do some more work here. In the previous example, you have to "imbue" each stream with a locale that knows how to format your Money.

This might look like:

```
std::locale US_Locale(...);
is.imbue(US_Locale);
std::locale Fr_Locale(...);
os.imbue(Fr_Locale);
```

Dave: The "...": Is this an exercise left to the reader?

Howard: Always hated those! A locale consists of a collection of facets. Each facet has a specialized responsibility. One facet knows how to format money. We need to create a locale with a money-formatting-facet that knows how we want our Money formatted. Here is how you might write a French franc formatter:

```
class Fr_Moneypunct : public std::moneypunct<char, false>
{
protected:
  virtual char do_decimal_point() const {return ',';}
  virtual char do_thousands_sep() const {return '.';}
  virtual std::string do_curr_symbol() const
                                        {return "Franc";}
  virtual pattern do_pos_format() const {
   pattern result = {{char(sign), char(value),
char(space),
char(symbol)}};
   return result;
  }
  virtual pattern do_neg_format() const {
   pattern result = {{char(sign), char(value),
char(space),
char(symbol)}};
   return result;
  }
};
```

Then you can install it in a locale with:

```
std::locale Fr_Locale(std::locale(), new Fr_Moneypunct);
```

That ought to do it. Ok, I know it may look a little unfamiliar, but you've got to admit, it's not much code.

Dave: Agreed. How about a little more detail?

Howard: The base class std::moneypunct is a standard facet. Its default behavior is to format US currency. However all of this behavior lives under virtual methods that are overridable. The behaviors which are most likely to be changed, are the most easily overridable. For example, changing the decimal point character to ',' is demonstrated above with the do_decimal_point method.

The do_pos_format and do_neg_format methods have the responsibility for dictating the relative position of the sign, value and currency symbols. The default order is "symbol, sign, none, value". A "none" means that there may be zero or more spaces. A "space" means that there are one or more spaces. In our example, it was not enough to change the currency symbol from "$" to "Franc", we also wanted it to be printed after the amount, instead of before. That is why we changed the pattern order to "sign, value, space, symbol".

There is a do_negative_sign() method that returns a string. The first character of this string gets put in the "sign" spot designated by the pattern returned from do_neg_format(). If the string has more than one character, the remaining characters get put at the end of the entire formatting sequence. Thus if do_negative_sign() returns "()", then negative amounts can be formatted as:

```
(10.000,00 Franc)
```

Dave: There's that 10.000,00 Franc again. Where does the exchange rate go in?

Howard: Let's take a closer look at our Money class. We could put the exchange logic as a method in there with a method called us_to_france:

```
class Money
{
public:
Money(double amount) : amount_(100*amount) {}
Money& us_to_france() {amount_ /= .1685; return *this;}
double to_double() const {return amount_;}
private:
double amount_;
};
```

So now, assuming you had read in US $10,000.00, you might print a French version like:

```
os << amount.us_to_france();
```

and "59.347,18 Franc" will be printed.

I'm going to throw my standard disclaimer in here: The Money class presented here is just to demo locale. I really haven't put any serious design thought into Money itself.

Dave: Ok, so we've created a customized locale, with a customized facet. How does our Money class make use of this machinery?

Howard: The work we've done so far will affect both input and output. To keep this discussion short, let's just look at the output routine for our example Money class:

```
std::ostream&
operator<< (std::ostream& os, const Money& amount)
{
using namespace std;
ostream::sentry ok(os);
if (ok)
{
   const money_put<char>& fmt = use_facet<money_put<char> >
(os.getloc());
   ios_base::fmtflags saveflags = os.flags();
   showbase(os);
   if (fmt.put(os, false, os, os.fill(),
amount.to_double()).failed())
      os.setstate(ios_base::badbit);
   os.flags(saveflags);
}
return os;
}
```

The bad news is that this code may look strange to you. The good news is that it is pretty much boiler plate code. Most locale-savey output routines you write will look similar to this.

The first line with the sentry animal prepares the stream for output, and ensures that the stream is in good shape. Next the ostream is queried for its locale (which you earlier imbued). Then the locale is searched for its money_put facet via the global method use_facet. The money_put facet is a standard facet that will use your Fr_Moneypunct class through the magic of virtual functions. Finally, you simply ask the money_put facet to "put" your Money on the ostream. money_put knows how to handle amounts expressed as doubles or strings, so you must be able to convert to one of those. That is why the to_double method has been part of the Money class from the beginning.

Note that I've specialized Money to only output to narrow (char) streams. We could have easily templated everything on the character type in order to handle both narrow and wide streams. This is in fact what the standard library does.

Dave: Can you give me some other possibilities for locale?

Howard: I think space will prohibit us from discussing all the facets in this much detail, but I'll try to hit some highlights.

A collate facet is available for sorting characters. The default behavior is the case sensitive stuff you've come to know and love.

A collection of ctype facets wrap the functionality found in the C header <ctype.h> (and <wctype.h>).

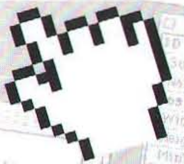
Numeric facets control the formatting of both integers and floating point numbers (input and output). There are several similarities here with the monetary facets described above, including the ability to specify decimal point characters and thousands seperators.

A collection of time facets are available to help you format times and dates. If you need to read or write "janvier" instead of "January", you can do it here. CodeWarrior offers a timepunct facet similar to monepunct and numpunct. This facet is non-standard, but so common as to have become a defacto standard. This makes it much easier to customize the names of the days of the week, or the names of the months.

Dave: locale sounds both powerful and flexible. But I can still safely ignore it if I want to right?

Howard: Yes, the default locale will give you the traditional formatting that we are all accustomed to.

Create REAL applications with one finger...



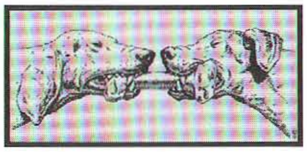
Use the others for something else.

REALbasic™

Visit **www.realbasic.com** for a FREE DEMO

REALbasic is a trademark of REAL Software, Inc.

Object-Oriented BASIC
100% Visual Interface Builder
Compile Native 680x0, PowerPC, and JAVA
Supports XCMDs, AppleScript and Native Plug-ins

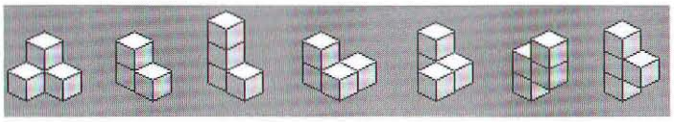


by Bob Boonstra, Westford, MA

Block Buster

One of the Vice Presidents at the Real Job (we'll call him Don) has a collection of unusual puzzles at his conference table. The kind that require you to move a large block through a small hole, untangle a set of intertwined metal loops, do the impossible with rope, or otherwise do something that requires mastery of the fourth spatial dimension or a greater aptitude toward right-brain thinking than I possess. Don retired this week, and, in his honor, this month's Challenge is devoted to a spatial reasoning exercise. Not a puzzle that would have been worthy of his conference table, perhaps, but something in that same spirit.

Our puzzle is a variation of the Soma Cube, reportedly conceived by a writer from Denmark named Peter Hein as he was listening to a lecture on quantum physics. (That's one class I don't remember being able to daydream through!) The object of the Soma Cube is to form a 3x3 cube using seven shapes formed from three or four of the smaller cubes:



Of course, the Soma can be assembled into other shapes besides a cube, which forms the basis for our Challenge. Your job will be to write code that will assemble a larger number of potentially larger shapes into an even larger designated shape.

The prototype for the code you should write is:

```
#if defined(__cplusplus)
extern "C" {
#endif

#define kUnknown 0xFFFF

typedef struct Cubie {
  SInt16 xCoord;       /* x coordinate of cubie */
  SInt16 yCoord;       /* y coordinate of cubie */
  SInt16 zCoord;       /* z coordinate of cubie */
  UInt16 value;        /* ordinal value assigned to the Piece this cubie is part of */
} Cubie;

typedef struct Piece {
  UInt32 numCubies;    /* number of individual cubies in this piece */
  Cubie  **theCubies;  /* pointer to array of cubies in this piece */
} Piece;

Boolean BlockBuster(
  /* return TRUE if you were able to solve the puzzle */
  long numPieces,      /* number of pieces to assemble */
  Piece thePieces[],   /* the pieces to assemble */
  Piece theGoal        /* the structure you should assemble */
    /* set (*theGoal.theCubies)[i].value to ((*thePiece->theCubies)+j)->value
       if piece j occupies cubie i in the reassembled puzzle */
);

#if defined(__cplusplus)
extern "C" {
#endif
```

The building block for our puzzle is the Piece, which is formed from numCubies individual cubes, provided to you in an array pointed to by theCubies. Each Piece has a unique nonzero identifier, which is associated with the value field in the Cubie structure. Cubies also have x, y, and z coordinates that define their relative orientation toward one another. There are no restrictions on the size or shape of a Piece, except that the constituent Cubies will all be connected (i.e., adjacent to another Cubie in the same Piece in x, y, or z.

Your BlockBuster routine is given an array (thePieces) of numPieces Pieces to work with. It is also provided with theGoal, an assembly of Cubies that you are to create from thePieces. For convenience, theGoal is also described using the Piece data structure. On input, the occupied coordinates are assigned a value of kUnknown. On output you should replace that value with the value of the Piece that occupies that coordinate. You may rotate and translate thePieces as you desire, but you may not break them by changing the relative orientation of the Cubies. You may use each Piece only once in assembling theGoal shape.

All puzzles will be solvable, but if you feel you cannot solve a puzzle, BlockBuster should return FALSE.

As an example, the Pieces in the standard Soma Cube might be described as follows (with each 4-tuple representing the x, y, z, and value components of a cubie:

```
{0,0,1, 5}, {1,0,1, 5}, {0,1,1, 5}, {0,0,0, 5},
{0,0,0, 1}, {1,0,0, 1}, {0,1,0, 1},
{0,0,0, 3}, {1,0,0, 3}, {2,0,0, 3}, {0,1,0, 3},
{0,0,0, 7}, {1,1,1, 7}, {0,1,0, 7}, {0,1,1, 7},
{0,0,0, 6}, {1,0,0, 6}, {0,1,0, 6}, {0,1,1, 6},
{0,0,0, 4}, {1,0,0, 4}, {2,1,0, 4}, {1,1,0, 4},
{0,0,0, 2}, {1,0,0, 2}, {2,0,0, 2}, {1,1,0, 2},
```

The winner will be the solution that assembles theGoal shape in the minimum amount of time. There are no storage constraints for this Challenge, except that it must execute on my 96MB 8500/200.

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

THREE MONTHS AGO WINNER

Sometimes Challenge solutions are very close to one another in performance, and other times the margin of victory is overwhelming. This month the winner falls into the latter category. Congratulations to **Ernst Munter** (Kanata, Ontario) for submitting an entry that soundly defeated the other four Challengers in the May Boggle contest. The objective was to score the most points by rearranging a Boggle® puzzle to form high scoring words, while at the same time minimizing the time penalty imposed for execution time. While Ernst took an order of magnitude more time to generate a solution, he found nearly four times as many unique words as his closest competitor, resulting in by far the highest score.

As you might suspect by looking at relative execution times, Ernst's strategy was a comparatively sophisticated one that he calls "simulated annealing". He begins by eliminating from the dictionary all words that cannot be formed by the letters available in the given instance of the puzzle. He then computes the frequency with which possible letter pairs occur in the dictionary, and uses this computation to exhaustively rearrange the puzzle letters and maximize what he calls "energy", the sum of the weights of the letter pairs in the rearranged puzzle. Only then does Ernst identify which dictionary words are present in the puzzle and compute a score. Optimization is done by randomly exchanging letter pairs until the expected gain for continuing exceeds the estimated penalty for continuing, after which the solution terminates.

One of the other solutions started by finding the longest (and therefore highest scoring) word in the dictionary that could be formed by the given letters and rearranging the puzzle to form that word. Another solution divided the board squares into three categories: corners, edges, and middle squares. It then rearranged the board so that "preferred" letters, determined a priori, were moved to the middle, and unfavored letters were moved to the corners.

There were 24 test cases used to evaluate this Challenge, 6 cases for each puzzle size (4, 5, 6, and 7). The table below lists, for each of the five solutions submitted, the cumulative number of words found, the number of those words that were unique, the associated points earned for the unique words, the time penalty (in points), and the net score. It also includes code size, data size, and programming language used. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges to date prior to this one.

Name Lang.	Total Words	Unique Words	Points	Time Penalty	Score	Code	Data
Ernst Munter (364) C++	11560	11560	39932	5191	34741	5840	8
JG Heithcock (10) C	5055	3037	3823	7	3816	1688	148
Tim Gogolin C	4121	2368	3275	47	3228	3456	412
Randy Boring (77) C	3154	1976	2828	4	2824	2836	944
Eric Hangstefer (9) C	1858	1858	2594	477	2117	3832	68

TOP CONTESTANTS

Here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated more than 10 points during the past two years. The numbers

StoneTable

You thought it was **just** a replacement for the List Manager ?

We lied, it is **much** more !

Tired of always adding just one more feature to your LDEF or table code ? What do you need in your table ?

Pictures and Icons and Checkboxes ?
adjustable columns or rows ?
Titles for columns or rows ?
In-line editing of cell text ?
More than 32K of data ?
Color and styles ?
Sorting ?
More ??

How much longer does the list need to be to make it worth $200 of your time ?
See just how long the list is for StoneTable.

Make StoneTable part of your toolbox today !
Only $200.00 MasterCard & Visa accepted.

StoneTablet Publishing
More Info & demo Voice/FAX (503) 287-3424
http://www.teleport.com/~stack stack@teleport.com

below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	210	10.	Murphy, ACC	24
2.	Boring, Randy	74	11.	Hart, Alan	21
3.	Cooper, Greg	54	12.	Antoniewicz, Andy	20
4.	Mallett, Jeff	50	13.	Day, Mark	20
5.	Rieken, Willeke	47	14.	Heithcock, JG	20
6.	Nicolle, Ludovic	34	15.	Higgins, Charles	20
7.	Lewis, Peter	31	16.	Hostetter, Mat	20
8.	Maurer, Sebastian	30	17.	Studer, Thomas	20
9.	Gregg, Xan	24			

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points	5th place	2 points
2nd place	10 points	finding bug	2 points
3rd place	7 points	suggesting Challenge	2 points
4th place	4 points		

Here is Ernst's winning solution to the Boggle Challenge:

BOGGLE.CP

Copyright © 1998,
Ernst Munter

```
/*
The Problem
------
A boggle board (Boggle®, a Parker Brothers game) shows a 2-D array of
letters. The task is to find as many words as possible which can be
traced on the puzzle, by going from letter to adjacent letter without
using the same letter twice. Actually, it is the point count that
should be maximized, where longer words count significantly higher
than short words.

As a variant of the original game, an extra degree of freedom is available:
the board may be rearranged to maximize the point count.

Solution Strategy
---------
Finding all words that can be made on the given puzzle is fairly straight
forward, with a scan of the dictionary. The real challenge is to find a
way to rearrange the board efficiently. This seems to be a very hard
problem to do exactly, that is to find the absolutely best arrangement.

I settled for a Monte Carlo approach, and try random shuffles until a
good score is reached. The result is likely far from the optimum.

My strategy has a number of components.

First, I extract from the dictionary all candidate words for which the
letters in the puzzle suffice, regardless of position.

Then I try to pre-arrange the puzzle into a state where letters that are
adjacent in the candidate words are as much as possible adjacent also
in the puzzle. To this end, I use a notion of "energy": Each possible
2-letter combination is given an energy value, equal to the sum of all
candidate word values which contain this letter pair.

We can now compute a total energy for the boggle board which is maximized
by a greedy algorithm as follows:

- The energy of each letter position is computed as the sum of the energies
  of all pairings with the adjacent letters.
- At each step, all letter pair exchanges are evaluated to see which would
  result in the largest increase in puzzle energy.
- If no increase is possible we are done.
- The selected pair is exchanged, and the previous steps are repeated.

The next phase is to evaluate the puzzle by finding all (candidate) words
that match the board in its current state.

The score is then hopefully increased by a random search:

- One letter pair is exchanged at random,
- The puzzle is re-evaluated,
- The configuration with the higher score is kept.
- Try again, until we run out of time.

Time Management
---------
The point score will be diminished by 1 point for each 20msec of running
time. Rather than run for an arbitrary amount of time, or an fixed number
of random swaps, an adaptive algorithm is used:

The elapsed time is measured after each run of random swaps, to establish
an average cost for each swap, which includes the time to evaluate its
effect on points.

The first run is for a fixed number of 32 swaps, to establish a baseline.
Subsequent runs are limited in length by the following consideration.
Using the known average time per run, I can fairly accurately predict the
number of points to be lost by a run of N swaps. And I might gain some
speculative number of points due to one or more lucky swaps.

As long as the expected gain exceeds the expected risk, we may continue
to try random swaps. But it might be wise to limit the potential loss
in each run to a small number such as 5 or 10% of the total points.
Larger boards result in larger variations, and I take a larger risk,
hopefully for a larger gain.

Assumptions
------
Memory required is for the Puzzle Structure (about 7KB) plus 12 bytes
per "candidate", that is words which potentially fit the current puzzle.

No assumptions are made of persistence of the private storage between
calls. Each call is handled completely independently. One could
remember the puzzle size, and if the same, avoid recomputing the
puzzle Neighborhood structure, but this is hardly worth the trouble.
*/

#include "boggle.h"
#include <string.h>
#include <timer.h>

enum { kMinLen=3,                    // no points for words < kMinLen
      kMaxDim=7,
      kMaxDimSquare=kMaxDim*kMaxDim,
      kHuge=127};

enum { kPenaltyRate=20000,  // 20msec per point
      kFirstRun=32,
      kDefRiskFactor=13};           // 12 or 13 seem to work best

struct Timer {                      // microseconds system timer
  int T0;
  void StartTimer() {
    UnsignedWide UWT;
    Microseconds(&UWT);
    T0=UWT.lo;
  }
  int ReadTimer(){
    UnsignedWide UWT;
    Microseconds(&UWT);
    return UWT.lo-T0;
  }
};

GetProfile
// A word profile is a 32-bit bit map of letters used in word
static unsigned long GetProfile(const char* word,int num) {
  unsigned long p=0;
  for (int i=0;i<num;i++) {
    p |= 1<<(*word++ & 31);
  }
  return p;
}
```

```
static unsigned long GetProfile(const char* word) {
  int c;
  unsigned long p=0;
  while (0 != (c=*word++)) {
    p |= 1<<(c & 31);
  }
  return p;
}
```

Candidate

```
struct Candidate {// A dictionary word that could fit
  const char*       word;
  unsigned long     profile;
  unsigned short    value;
  unsigned char     fit;
  unsigned char     tempFit;
  Candidate(){};

  Candidate(int len,const char* wd){
    word=wd;
    profile=GetProfile(wd);
    switch(len){
case 0:
case 1:
case 2:   value=0;break;
case 3:
case 4:   value=1;break;
case 5:   value=2;break;
case 6:   value=3;break;
default:value=5+6*(len-7);
    }
    fit=tempFit=0;
  }
};
```

Neighbors

```
struct Neighbors {
  char* next[9];  // pointers to the 8 neighbors of
};                           // a puzzle field, plus a NULL

enum {MIDDLE,EDGE,CORNER};
struct Neighborhood {
  Neighbors nb[kMaxDimSquare]; // 1 set per puzzle field
  void Init(int dim,char* c) {
    int dimSq=dim*dim;
    int col,row,x=0;
    Setup(x++,c++,1,dim,CORNER);
    for (col=1;col<dim-1;col++) Setup(x++,c++,1,dim,EDGE);
    Setup(x++,c++,-1,dim,CORNER);
    for (row=1;row<dim-1;row++) {
      Setup(x++,c++,dim,1,EDGE);
      for (col=1;col<dim-1;col++)
Setup(x++,c++,1,dim,MIDDLE);
      Setup(x++,c++,dim,-1,EDGE);
    }
    Setup(x++,c++,1,-dim,CORNER);
    for (col=1;col<dim-1;col++) Setup(x++,c++,1,-dim,EDGE);
    Setup(x++,c++,-1,-dim,CORNER);
  }
  void Setup(int from,char* c,int A,int B,int type) {
    char** np=nb[from].next;
    switch (type) {
case MIDDLE:
    *np++=c-A-B;
    *np++=c-B;
    *np++=c+A-B;
case EDGE:
    *np++=c-A;
    *np++=c-A+B;
case CORNER:
    *np++=c+A;
    *np++=c+B;
    *np++=c+A+B;
    *np=0;
    }
  }
};
```

Tour

```
struct Tour { // Sequence of puzzle locations
```

CALLING ALL CODE WARRIORS!

Winning the battle but losing the war…?

Synergex's plug-in to Metrowerks' CodeWarrior lets you control complex software projects from the comfort of your favorite IDE!

INTERSOLV PVCS Version Manager™ for Macintosh®

- Manage versions of any file type—even HTML!
- Check files in and out without leaving CodeWarrior
- Use the industry's top-rated multi-platform SCM tool

TO ORDER CALL: 800-366-3472

www.synergex.com

pvcs-info@synergex.com

2330 Gold Meadow Way – Gold River, CA 95670

SYNERGEX *Take Part In Creating Success* INTERSOLV

All trademarks are the property of their respective owners.

```
  char* tour[1+kMaxDimSquare];
  void Clear(){tour[0]=0;}
  void Add(char* loc) {
    char** tp=tour;
    while (*tp) tp++;
    *tp=loc;
    *++tp=0;
  }
  void Replace(char* loc1,char* loc2) {
    char** tp=tour;
    while (*tp) {
      if (*tp==loc1) {
        *tp=loc2;
        return;
      }
      tp++;
    }
    return;
  }
};
```

Stack

```
struct Stack {  // used to remember the track when
  char** CP;     // tracing a word through the puzzle
  char val;
  void Push(char** c,char v){CP=c;val=v;}
  void Pop(char** & c,char & v){c=CP;v=val;}
};
```

Puzzle

```
struct Puzzle {
  int      dimension;          // 4 to 7
  int      dimSquare;          // 16 to 49
  int      points;             // computed points
  int      riskFactor;
  unsigned long profile; // of letters in puzzle
  Stack*   SP;
  char*    P;                  // ref to the 'puzzle'
  Timer     tmr;
  char     letterCount[32];
  int      energy[kMaxDimSquare]; // energy of puzzle fields
  Stack  stack[kMaxDimSquare];
  Tour   TOUR[32];                // list of puzzle fields
  Neighborhood N;
  int    pairs[32*32];            // values of letter pairs

Puzzle::Init
  void   Init(int dim,char* puzzle) {
    dimension=dim;
    dimSquare=dim*dim;
    points=0;
    SP=stack;
    P=puzzle;
    memset(letterCount,0,32);
    for (int i=0;i<32;i++) TOUR[i].Clear();
    for (int i=0;i<dimSquare;i++) {
      char c=P[i];
      letterCount[c&31]++;
      TOUR[c&31].Add(P+i);
    }
    profile=GetProfile(P,dimSquare);
    N.Init(dimension,P);
    riskFactor=kDefRiskFactor-dim;   // more risk for larger puzzles
  }

Puzzle::Trace
  int Trace(Candidate* cd) { // trace 1 word
    const char* wd=cd->word;
    char** tour=TOUR[*wd & 31].tour;
    char* loc=*tour;
  char letter;

  // loop unrolled for the first and second letters of the word
first:
    letter=*loc;
      if (letter=='Q') ++wd;    //skip 'U' after 'Q'

  // deal with the first and second letters
      SP++->Push(tour,letter);
      *loc=0;              //hide 1st letter
      tour=N.nb[loc-P].next;    //goto 2nd tour
      loc=*tour;  ++wd;
second:
      letter=*loc;
      if (letter==*wd) {
    if (letter=='Q') ++wd;    //skip 'U' after 'Q'
    if (wd[1]==0) goto success;
    SP++->Push(tour,letter);
    *loc=0;                   //hide 2nd letter
    tour=N.nb[loc-P].next;
    loc=*tour; ++wd;
    goto third;               //goto 3rd tour
      } else {
        loc=*++tour;          // next 2nd loc
    if (loc) goto second;

    (-SP)->Pop(tour,letter);
    **tour=letter;            //restore first letter
    if (letter=='Q') -wd;
    -wd;
    loc=*++tour;
    if (loc) goto first;
      }
      return 0;              //fail

third:
    do {
      letter=*loc;
      if (letter==*wd) {
    if (letter=='Q') ++wd;    //skip 'U' after 'Q'
    if (wd[1]==0) break;      //success
    SP++->Push(tour,letter);
    *loc=0;              //hide this letter
    tour=N.nb[loc-P].next;
    loc=*tour; ++wd;
      } else do {
    loc=*++tour;
    if (loc==0) {
      if (SP<=stack) {
        return 0;
      }
      (-SP)->Pop(tour,letter);
      **tour=letter;          //restore last hidden letter
      if (letter=='Q') -wd;
```

```
          -wd;
      } else break;
        } while(1);
      } while(1);
success:
    while (SP>stack) {        // restore all letters
      (-SP)->Pop(tour,letter);
    **tour=letter;
    loc=*tour;
    }
    return 1;
  }

Puzzle::IsPossible
  int IsPossible(const char* word) {// all letters present
    int c,len=0;
    char myDist[32];
    memset(myDist,0,32);
    while (0 != (c=31 & *word++)) {
      if (letterCount[c]>myDist[c]) {
        myDist[c]++;
        len++;
      } else return 0;
      if ((c==(31 & 'Q')) && (*word))
        word++;                    // don't need U after Q
    }
    return len;
  }

Puzzle::Swap
  void Swap(const int a,const int b){
    char ca=P[a],cb=P[b];
    TOUR[ca&31].Replace(P+a,P+b);
    TOUR[cb&31].Replace(P+b,P+a);
    P[a]=cb;
    P[b]=ca;
  }

//The next set of functions are for points maximization

Puzzle::RandomShake
// RandomShake just swaps two letters in the puzzle,
// making sure they are different, and returns the
// profile of the two letters
  unsigned long RandomShake(int & a,int & b) {
    char ca,cb;
    do {
      a=rand()%dimSquare;
      b=rand()%dimSquare;
    } while ((a==b) || ((ca=P[a])==(cb=P[b])));
    TOUR[ca&31].Replace(P+a,P+b);
    TOUR[cb&31].Replace(P+b,P+a);
    P[a]=cb;
    P[b]=ca;
    unsigned long shakeProfile=(1L<<(ca&31)) | (1L<<(cb&31));
    return shakeProfile;
  }

Puzzle::Evaluate
//The function Evaluate checks all candidates for fit,
// and computes the total point score for the puzzle
  void Evaluate(Candidate* cd,int numCandidates) {
    int pts=0;
    for (int i=0;i<numCandidates;i++) {
      if (0 != (cd->fit=Trace(cd))) {
    pts+=cd->value;
      }
      cd++;
    }
    points=pts;
  }

Puzzle::Improves
// Improves() answers the question: would the proposed shakup
// increase the score. Computes candidates.tempFit.
// Uses shakeProfile to avoid recomputing candidates that
// are unaffected by the swap.
  bool Improves(
  unsigned long shakeProfile,
  Candidate* cd,int numCandidates) {
    int pts=points;
    for (int i=0;i<numCandidates;i++) {
      if ((shakeProfile & cd->profile)) {
        cd->tempFit=Trace(cd);
        pts+=(cd->tempFit-cd->fit)*cd->value;
      } else {
        cd->tempFit=cd->fit;
      }
      cd++;
    }
    if (pts>points) {
      points=pts;
      return true;
    }
    return false;
  }
```

`Puzzle::UpdateFits`
// UpdateFits commits tempFit to fit, after a swap is accepted

```
  void UpdateFits(Candidate* cd,int numCandidates) {
    for (int i=0;i<numCandidates;i++) {
      cd->fit=cd->tempFit;
      cd++;
    }
  }
```

`Puzzle::Rearrange`
// Rearrange shakes the puzzle a specified number of times
// and retains the configuration yielding the highest score

```
  void Rearrange(int howOften,Candidate* cd,int numCandidates)
{
    for (int i=0;i<howOften;i++) {
      int s1,s2;
      unsigned long shakeProfile=RandomShake(s1,s2);
      if (Improves(shakeProfile,cd,numCandidates)) {
        UpdateFits(cd,numCandidates);
      } else {
        Swap(s1,s2);
      }
    }
  }
```

`Puzzle::Optimize`
// Optimize calls Rearrange a number of times, to strike a
// compromise between points gained by random shakes, and
// points lost due to runtime cost.

```
  void Optimize(Candidate* cd,int numCandidates) {
    tmr.StartTimer();
    int run=kFirstRun;
    int countRuns=0;
    int runningRate;
    int oldPoints=points;
    int basePoints=points;
    Rearrange(run,cd,numCandidates);
    int runTime=tmr.ReadTimer();
    int lost=(runTime+kPenaltyRate/2)/kPenaltyRate;
    do {
      oldPoints=points;
      countRuns+=run;
      runningRate=runTime/countRuns;
      int pointsToRisk=points/riskFactor-lost;
      int run=kPenaltyRate*pointsToRisk/runningRate;
      if (run<1) break;
      Rearrange(run,cd,numCandidates);
      runTime=tmr.ReadTimer();
      lost=(runTime+kPenaltyRate/2)/kPenaltyRate;
    } while (points > oldPoints);
  }
```

//The next set of functions relate to energy maximization

`Puzzle::PrimePairs`
// PrimePairs uses letter pairs in all candidate words
// to construct an array of basic energy values for each.

```
  void PrimePairs(Candidate* cd,int numCandidates) {
    memset(pairs,0,sizeof(pairs));
    for (int i=0;i<numCandidates;i++) {
      const char* wd=cd->word;
      int a=*wd & 31,b;
      do {
```

Build great applications.
Better. Cheaper. Faster!

CodeWarrior
Symantec
MacOS 8
System 7
68K
PowerPC
C/C++
Pascal

Good, fast and cheap.
***They* say pick two. *We* say you can have it all.**

Tools Plus routines let you create user interface elements that work as soon as you make them. They're ready-to-use Macintosh application parts, just waiting to be assembled.

We include everything from tool bars and floating palettes to the coolest picture buttons and tons of 3D controls. **Tools Plus** works brilliantly with multiple complex dialogs, using resources and/or manual coding.

Use **Tools Plus** libraries alone or with our framework. It's compact yet immensely capable. Easy to learn, yet ready to grow with you.

Good, Fast and Cheap.
Don't get two out of three.
Get Tools Plus.

Tools Plus for

CodeWarrior Pro and Gold (68K and PPC)	$249
CodeWarrior Bronze (68K)	$199
Symantec Compilers (C/C++ & Pascal, 68K and PPC)	$249
Symantec (THINK) C/C++ (68K)	$149
THINK Pascal (68K)	$149
THINK C/C++ & THINK Pascal (68K)	$199

Compact | Fast | Easy

Tools Plus
LIBRARIES + FRAMEWORK

NEW TOOLS PLUS 4.0
MacOS 8
APPEARANCE SAVVY

"All in all, it's an incredibly rich collection of tools…If you are interested in developing applications that have 'quality' written all over them, then Tools Plus is for you."
MacTech MAGAZINE

Finalist Macworld 1997 Editors' Choice Awards
Best Rapid-Development Tool

"…the routines are more compact and faster than anything you might write…Every element of Tools Plus is useful…a bargain compared with coding these routines yourself."
MW ★★★★
Macworld

Orders and Enquiries:
Phone: (416)219-5628
Fax: (905)847-1638
WaterEdgSW@aol.com

Free Evaluation Kit:
Available at our web site
http://www.interlog.com/~wateredg

Water's Edge Software
2441 Lakeshore Rd W.
Box 70022
Oakville, Ontario
Canada, L6L 6M9

```
        b=*++wd & 31;
        if (b==0) break;
        pairs[32*a+b]=pairs[a+32*b]+=cd->value;
        a=b;
      } while(1);
      cd++;
    }
  }
```

Puzzle::InitEnergy
```
// Computes the energy of each puzzle field
  void InitEnergy() {
    for (int i=0;i<dimSquare;i++)
      energy[i]=GetEnergy(i);
  }
```

Puzzle::GetEnergy
```
// Get the energy for one field, using the Neighborhood
// struct to trace the field's neighbors.
  int GetEnergy(int k) {
    char** tour=N.nb[k].next;
    char*  loc;
    int* pairP=pairs+32*(P[k]&31);
    int e=0;
    while (0 != (loc=*tour++)) {
      e+=pairP[*loc&31];
    }
    return e;
  }
```

Puzzle::CommitSwap
```
// This swap did some good, let's keep it
  void CommitSwap(int a,int b){
    Swap(a,b);
    energy[a]=GetEnergy(a);
    energy[b]=GetEnergy(b);
  }
```

Puzzle::ComputeDelta
```
// ComputeDelta makes a tentative swap and returns
// the increase or decrease in energy.
  int ComputeDelta(int a,int b) {
    if ((a==b)||(P[a]==P[b])) return 0;
    Swap(a,b);
    int ea=GetEnergy(a);
    int eb=GetEnergy(b);
    Swap(a,b);                    // restore previous state
    return ea+eb-energy[a]-energy[b];
  }
```

Puzzle::ComputeBestSwap
```
// ComputeBestSwap tries all possible swaps and returns
// the swap yielding the highest energy increase
  int ComputeBestSwap(int & aa,int & bb) {
    int bestDelta=-10000;
    for (int a=1;a<dimSquare;a++) {
      for (int b=0;b<a;b++) {
        if (P[a]^P[b]) {
          int delta=ComputeDelta(a,b);
          if (delta>bestDelta) {
            bestDelta=delta;aa=a;bb=b;
          }
        }
      }
    }
    return bestDelta;
  }
```

Puzzle::MaximizeEnergy
```
// MaximizeEnergy keeps swapping puzzle letters until no
// more increase in puzzle energy can be obtained.
  void MaximizeEnergy() {
    int a,b,delta=ComputeBestSwap(a,b);
    do {
      if (delta>0) CommitSwap(a,b);
      else break;
      delta=ComputeBestSwap(a,b);
    } while (1);
  }
};
```

PrivateData
```
struct PrivateData { // just the puzzle and the candidate words
  Puzzle pzl;
  int    numCandidates;
  Candidate candidates[1];       // open ended array

  void Init(int dimension,char* puzzle) {
    pzl.Init(dimension,puzzle);
  }

  void SelectCandidates(int dictSize,const char *dictionary[])
{
    numCandidates=0;
    for (int i=0;i<dictSize;i++) {
      const char* word=dictionary[i];
      int len=pzl.IsPossible(word);
      if (len>=kMinLen)
        candidates[numCandidates++]=Candidate(len,word);
    }
    pzl.PrimePairs(candidates,numCandidates);
  }

  void Solve() {
    pzl.InitEnergy();
    pzl.MaximizeEnergy();
    pzl.Evaluate(candidates,numCandidates);
    pzl.Optimize(candidates,numCandidates);
  }

  int CopyResults(const char *wordsFound[]) {
    int n=0;
    Candidate* cd=candidates;
    for (int i=0;i<numCandidates;i++) {
      wordsFound[n]=cd->word;
      n+=cd->fit;
      cd++;
    }
    return n;
  }
};
```

Boggle
```
// the function Boggle is published in boggle.h
long Boggle(
  long dimension,
  char puzzle[],
  long dictSize,
  const char *dictionary[],
  const char *wordsFound[],
  void *privStorage
  ) {

  int nFound=0;
  srand(10001); // just to make it repeatable

  if ((unsigned long)dimension <= kMaxDim) {
  PrivateData* PD=(PrivateData*)privStorage; // assumed to be enough
    PD->Init(dimension,puzzle);
    PD->SelectCandidates(dictSize,dictionary);
    PD->Solve();
    nFound=PD->CopyResults(wordsFound);
  }

  return nFound;
}
```

MT

Want to suggest an article for MacTech Magazine?
Send your suggestions to <editorial@mactech.com>

by Steve and Patricia Sheets

Protect Your Software with Hardware: Adding Software Protection Systems to Your Programs

Hardware to protect your software

Software Piracy is a fact of life for Macintosh developers. For popular programs in the US, there are more illegal copies created than legal ones. The numbers are even worse internationally, where over 90% of software is pirated. The original method to combat this piracy was software copy protection. This has become a thing of the past. Modern users expect to be able to make archive copies of all software, while modern pirates can easily break any software based disk scheme. For most software companies, anti-piracy planning involves working with organizations like the Software Publishers Association and the Justice Department to stop institutionalized piracy. After that point, the losses are considered to be part of "doing business".

While piracy of the more main stream products is painful, when more expensive vertical applications (ones intended for small markets) are copied, it can break a small company. A developer that only expects to sell 500 copies of his program needs to do all he can to make sure no one steals even one customer. To combat theft, several manufactures have been creating hardware based anti-piracy solutions for these software developers. These hardware devices, commonly called "dongles", are required for the software to run correctly. While a pirate (or legitimate user) can easily make copies of the software product, he can not easily duplicate the accompanying hardware device. For the cost of the "dongle" (between $20 to $40 depending on features and quantities), a software developer has taken a major step to protect his valuable software package.

This article explains the ideas and advantages of using a hardware "dongle", as well as the effectiveness of these products. Other less know uses of "dongles" will be explained. Three of the main manufacturers of Macintosh "dongles", Aladdin Knowledge System, Rainbow Technologies and Micro Macro Technologies will be discussed. Each product features and APIs will be explained.

Apple Desktop Bus to the Rescue

In the Macintosh world, the preferred "dongle" type is an ADB device (Apple Desktop Bus). These 4-6 inch devices plug into the normal desktop bus chain of the Mac. The devices universally look like short cables with a fat connector on one end. Unlike similar PC devices that plug into a parallel or serial port, there is no conflict with the normal operation of the port. The Mac "dongles" do not monopolize a port, and the logical limit of 16 ADB devices usually is not a factor. There is an additional physical limit of how many "dongles" can be used at once, because the devices use the power of the ADB. The number of usable "dongles" is dependent on the keyboard, mouse and cables being used, but a developer can safely assume at least five "dongles" will function on one computer.

Since the "dongle" is an ADB device, it needs to be connected at boot time to function correctly. When the application runs, it communicates with the device to insure that the user is legitimate. This communication is secured, and nearly impossible to break. Developer places calls inside their program to make this communication. It is a good idea not to just query the "dongle" one time at startup, but to actually do the checking at various locations in the program. This way the user can not trick the program by detaching the "dongle", after the program has started, to be used on another machine. On the other hand, do not call the "dongle" every time through the main event loop. The "dongle" APIs are fast, but not that fast. A good rule of thumb is to put the call before any

Steve Sheets has been happily programming the Macintosh since 1983, which makes him older than he wishes, but not as young as he acts. **Patricia Sheets** is currently a free-lance writer, and is the more creative one of the family. Their wanderings have led them to Northern Virginia. For those interested, their non-computer interests involve their daughters (ages 2 & 10), Society for Creative Anachronism (Medieval Reenactment) and Martial Arts (Fencing, Tai Chi). Steve is currently an independent developer, taking on any, and all projects and can be reached at "MageSteve@AOL.Com".

IO portion of the program (Open File, Save File, Print), or at any time where different functions are invoked (spreadsheet mode vs. word processor). The more locations that call the API, the harder it is for software pirates to reverse engineer and modify the program to not use the "dongle". Some programmers notice that these are same locations they would normally modify when creating "crippled" shareware versions of the software vs. registered copies. Yes, the same ideas apply.

How effective is a "dongle"? If a software developer is looking for a perfect software piracy scheme, he will not find it here, and may never find it anywhere. The original software can be reversed engineered, and jump commands could be placed over the machine code that calls the various "dongle" APIs. While difficult to do the various algorithms built into the device, even the hardware itself could be reversed engineered and someone could design "pirate dongles". Either way, it would take a considerable investment of time and money for the pirate to crack the product. For most casual pirates, the protection provided by the "dongle" is fairly absolute. The user needs to have the device to run his program. The professional pirates would definitely steer away from the expense of duplicating "dongled" software. Mass pirating of the developers software would be dramatically decrease.

"Dongle" Feature Set

The three makers of "dongles" in this article use different methods to verify that the user is the correct one. However, all of them have common features and designs. When the developer purchases his "dongles" to be bundled with his software, he is given a single ID that uniquely identifies him. All the "dongles" he purchases will have this ID, as well as a unique ID for the given device. When the user's program is running, calls added to the program interrogate the ADB device, and look for these IDs. Generally, the Developer ID is all that is used to verify that the software has the correct "dongle' to run, but the programmer is not limited to this. A program can be written to use the device id to uniquely identify the program when accessing the Internet. It could be included when emailing registration information to the developer. For an in-house developer, the software itself could be recompiled any number of times, so each rev of the software requires one and only one unique "dongle" to function.

Several of the "dongles" provide more than just developer/device ID. Some provide temporary and permanent memory storage. The temporary storage is RAM on the device that the programmer can access. He can store any data he wished onto the RAM. No matter what the user does (quit the application, trash the System preference folder or delete the software application), this value will remain there until it is modified or the user shuts down the machine. The RAM might be used by different applications from the same developer as shared memory.

The permanent memory storage (often called Registers) of the "dongles" last even if the user disconnects the device from the computer. This is often implemented using ASIC (Application Specific Integrated Circuit). In fact, these Registers can be preset from the developer to a given value. The makers of the devices provide the developer with utility programs that allow them to modify these Registers before they ship them to their customers. Once in the users hands, the Registers can function as Read Only memory or Read/Write memory. For example, the applications serial number could be stored into a Register.

Some of the makers of "dongles" sell network versions of their application. These usually involve a special version of the "dongle" that needs to be connected to the network. When the application is run, it checks the network for the computer with the "dongle", and interrogates it just as if the "dongle" was directly connected to that machine. Network version of the "dongle" are very good for site licensed versions of the software. For example, a class room software might allow only 30 student to access the Exam program at the same time. To allow more users on the system, the school district must purchase an upgraded registration.

Uses for a "Dongle"

One of the most common uses of these Registers are as software counters. The developer sets a Register to a given value. Each time the program is booted, the counter is decremented by one. When it reaches zero, the program will no longer function. This is a guaranteed way to insure that "demo" software is not overused. The counter could increment the value, in which case the number provides untamperable statistics of usage (good for in-house applications).

In another use, if the owner purchases a large suite of applications, his "dongle" may be programmed to only allow him to use modules or programs A, D & F. The entire suite may be shipped to him on CD (showing him the features), but he can only run the parts he has paid for. The unpaid for modules might be unlocked for him, if he receives the correct registration value to enter into his program. This registration value is then moved into the "dongle", and will function even if he reinstalls the application. The Registration number would be based on his "dongled" id, thus would be unique to him, and unusable by someone with the same product.

This same scheme could also be used to register shareware versions of the software, and allow full use of the program. A software company makes and distributes free CDs with the full version of the software on it (including all manuals and needed files), but have it only function in "crippled" or shareware mode. When the user contacts the developer for a full version, he is sent a "dongle", personally identified to him that allows uses of all the features.

The Choices

The following is a description of three of the main "dongles" on the Macintosh market. Each company and product has it's advantages and disadvantages, but all have good functional products.

Aladdin Knowledge Systems "MacHasp"

Aladdin Knowledge Systems is the manufacturer of the MacHASP product. They can be reached at 800-223-4277 or on the web at <http://www.aks.com>. It should be noted that this is NOT Aladdin, makers of StuffIt. These are two different companies. Aladdin Knowledge Systems make one of the most highly rated "dongle" products.

The MacHASP product is split up into 4 different lines. The simplest product is the MacHASP-S. This is a barebones "dongle" containing the seed code/return code mechanism, a developer id, a device id, and two words of RAM storage. This product cost ranges from $29 (for 2-49 devices) to $21 (for 500-999 devices). For quantities above 1000 of any product, Aladdin should be contacted directly.

The MacHASP-M & MacHASP-M8 versions of the product contain all the features of the S version, along with permanent memory storage. The MacHASP-M device contains 90 bytes of Read/Write storage. The MacHASP-M8 device contains 986 bytes of Read/Write storage. Both devices contain an additional 10 bytes of Privileged memory. The MacHASP-M product cost ranges from $35 (for 2-49 devices) to $26 (for 500-999 devices), while the MacHASP-M8 version range is from $40 to $31 per unit.

In order for an application to access this memory, the program must have the Read or the Write passwords. If the developer only wants his program to Read this memory, he only places the Read password into the program. He uses the Write password with a provided utility (HASPEDIT) to modify the memory before shipping the HASP to his users. If he wants to modify the memory from within his program at the users site, then the Read & Write passwords are placed inside the program.

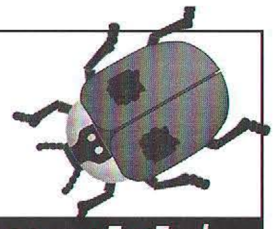
Because the permanent memory is divided into 2 sections (Read/Write & Privileged), the developer is actually given 4 passwords; a Read one, a Write one, a Privileged Read one, and a Privileged Write one. Thus some of the memory can be safely read and modified by a program (the normal Read/Write memory), while other section of it can be used as Read only (Privileged memory). Since the Privileged Write password is not imbedded inside the program, it is practically impossible to "crack".

The last product type available from Aladdin is the Net-MacHASP product. This consist of a single ADB "dongle" which is connected a single Macintosh running the Net-MacHASP server software. The developers software is linked with a library that allows the program to log onto the servers across the network. Using similar calls, the device can be interrogated just as if the HASP was connected to the Macintosh directly. This product can support different increment of stations (5, 10, 20, 50, 100, unlimited). The price for the product ranges from $83 to $279 dollars, depending on the number of devices purchased and the number of stations per device.

For those interested in using the product, Aladdin sells a very complete developer kit for $14.50. This kit includes 1 MacHASP device containing the "demo" developer id. The kit's CD provides interfaces for CodeWarrior (C & Pascal), Think C, MPW (C, Pascal & MrcC), Lingo, Hypercard, 4th Dimension, Serius, FoxBase and Excel. All the interfaces include source level examples. The documentation is easy to read and understand, even though there area a couple different manuals and addendums.

All of Aladdin's devices use a seed code/return code mechanism to securely verify the correct HASP is connected to the computer and software. The developer is provided with a seed code, which is used with the provided MacHASP routine to send a encrypted communication to the ADB device. If the Return code matches what the program expects, everything works correctly.

TestTrack



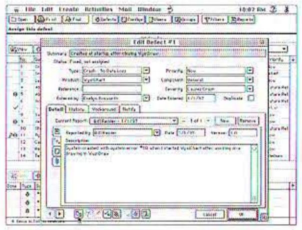
Your Total Bug Tracking Solution

"If you're a developer, you've got to get TestTrack. Your bugs will hate you for it!" *Cool Tool of the Day*

Discover the tool today's top software developers are using to improve the quality of their **Macintosh** and **Windows** applications—**TestTrack.**

- Track bugs, feature requests, problems, customer information, and more.
- New! E-mail notifications—SMTP and MAPI.
- New! Import bugs via e-mail automatically.
- Produce concise reports.
- Multiple users, full security–link engineers, testers, managers, even tech writers.
- New! Improved help desk support.
- New! Deferred defect numbering, external attachment storage, customer bug histories, and much, much, more.
- Automatically route bugs to team members.
- Track the history of each bug.
- Save time and improve tech support by giving Solo Bug, TestTrack's stand-alone bug reporter, to your customers.

Only $169! 2+ for $149 each!
To order call 888-683-6456
or 513-683-6456



Download our demo today!
http://www.seapine.com

sales@seapine.com
http://www.seapine.com

Seapine Software

Regardless of the computer language, all of the MacHASP functionality is through a single routine; MacHASP. Before this call can be made, a special code resource is required to either be linked into the program, or accessed from a separate Resource File (in which case, opening and closing the resource file must be done by the program). The recommended way to include the single "HASP" resource is to link it into the program. From C or Pascal, the MacHASPInit calls loads the resource and prepares the MacHASP routine to be used. The MacHASPClose call reverses this process. Once MacHASPInit has been called successfully, the MacHASP routine can be called.

The MacHASP routine passes a number of parameters including Service parameter (to indicate the requested action), multiuse input parameters and multiuse return parameters. The services available can be split into configuration information and read/write calls. The program can check how many hasps are connected the the ADB bus, find out the type of hasp connected (S, M, M8), and check the return code for the hasp using the particular seed id. Using the returned code, the program can verify the correct hasp is connected. At this point, the read/write calls can be used to examine or modify the RAM memory, permanent memory or privileged memory of the device. Remember, accessing anything other than the RAM requires a special password, thus locking out anyone except the correct program.

Aladdin has one important feature that needs to be mentioned; the MacHASP Envelope Utility. With the utility, the application file can be "wrapped" in a protective envelope keyed to a developer id. The program will not run, or even be accessed, unless the "dongle" is attached. However, unlike adding code to the software, this utility can be done to any software without accessing the source code. The conversion is simple and quick for any developer to do, and they have both 68K and PowerPC versions of the utility. Aladdin recommends using both API calls and using the envelope for the most protection.

Rainbow Technologies "SentinelEve3"

Rainbow Technologies is the manufacturer of the SentinelEve3 product. They can be reached at 800-852-8569 or on the web at <http://www.rainbow.com>. They are one of the largest manufacturers of "dongles" for the PC & Macintosh.

Rainbow's products are split up into 2 different lines. The stand alone device is the SentinelEve3. This device has 32 two byte General Purpose Registers (GPR), 4 two byte counters, 4 four byte algorithms and 4 bytes of RAM. The RAM memory is volatile, while all the rest is non-volatile. Developers can purchase the SentinelEve3 device for $24 (in batches of 100-499) to $22 (in batches of 500-1000). If larger batches are needed, contact Rainbow directly.

The network version of their product is called the NetSentinel-Mac. Again, a single network Macintosh is plugged into the "dongle". The user runs the developers program on another machine which does not have a "dongle". The program verifies whether or not it is allowed by communicating with the "dongled" Macintosh. From 1 to 50 users (set by the software developer) can communicate with one NetSentinel-Mac product this way. The price for this product ranges from $49 (100 to 499 units) to $44 (500 to 1000 units).

Rainbow sells 2 separate developer kits. The SentinelEve3 developer kits includes one "dongle", 2 floppies, a manual, and the needed developer ids and passwords. These are not demo ids, but the actual id that can be used by the finished software. The kit is available for $19.95. The kit's CD provides interfaces for CodeWarrior (C & Pascal), Think C, Think Pascal, MPW (C & Pascal), Hypercard, 4th Dimension, Serius, FoxBase, Excel, FoxPro, Future Basic, MacroMind Director, Microsoft Quick Basic, Omnis and SuperCard. All the interfaces include minimal examples. The NetSentinel-Mac version of the developer kit is similar, but sells for $24.95. This version of the developer kit includes the needed software for both the server and client sides of the product.

Rainbow provides a number of passwords, ids and algorithm numbers, that need to be understood. Each "dongle" contains a developer id (assigned by Rainbow) as well as a unique serial number. The developer also is given a developer password and a write password. The developer should only include the Write password in the shipping product when it needs to modify the memory. The developer id is used by the SentinelEve3 utility program to configure the device and should not be shipped with the product. The product also has 4 algorithm registers that the developer sets when the device is configured (basically seed values for the device's security). A program would query the algorithm to make sure the software is allowed to be run by the given SentinelEve3. Since there are 4 registers, four programs from the same developer could use same "dongle", or one program with four modules could be set up this way.

The GPR memory of the SentinelEve3 is divided up into a number of registers, and they are grouped into pairs. Each pair can be set to be read only or read/write using the SentinelEve3 utility program. While the registers can be used for any purpose, they can also be configured as challenge/response pairs. Using this scheme, the application sends the value contained in one register. If it is correct, the key returns the value in the other register. Since registers set this way can not be read directly, this provides a sophisticated method of verifying the "dongle".

The SentinelEve3 device contains 4 counter registers. They can be used to check how many times a program can be run. There are API calls to decrement the counter when needed. Or the counter can be tied into a algorithm number, so that the algorithm will fail when the counter has been set to zero. Again, the utility program is used to set the purpose and value of a counter.

The SentinelEve3 API is a standard library to be linked into the application program. The API is divided into 2 main sections, the normal mode and the developer mode. The developer mode is used strictly to configure the SentinelEve3 device. Most software developers will rarely program this mode directly, because the SentinelEve3 utility program provides them with all the capability they need. The normal mode calls are the routines that should be placed inside a developer's program.

Almost all of the calls to the SentinelEve3 device require a block of memory as a parameter . This memory is allocated by the RBEHANDLE call. This memory can be used over and over again by different calls. The RBEFINDFIRST and RBEFINDNEXT calls are used to search for a given SentinelEve3 device on the ADB. Given a developer id, it returns the serial number. The RBEQUERY call queries a specific algorithm register, while the RBEDNZ decrements a specific counter register. The RBEREAD and RBEWRITE routines can be used to examine and modify specific general purpose registers or RAM registers. The RBEWRITE call requires the Write password.

Micro Macro Technologies "MicroGuard"

Micro Macro Technologies is the manufacturer of the MicroGuard product. They can be reached at (732) 381-3042 or on the web at <http://www.micromacro.com>. They have one of the most versatile "dongles" around.

The MicroGuard product comes in 3 different configurations. The MicroGuard Plus is a full feature "dongle" containing 16 bytes of volatile RAM memory, 120 bytes of non-volatile memory, 40-bit encryption, 64 bit one way hash algorithm, 4 passwords and 64 bits of bit array. All save the RAM memory is non-volatile memory (exists even when power is shut off). The MicroGuard Plus M4 devices contain this, as well as an additional 496 bytes of extended non-volatile memory. Lastly, the MicroGuard Plus Net version is a fully network compatible version of the MicroGuard Plus device. The MicroGuard Plus price range is from $25 (1-99 units) to $23.75 (100-499 units). The MicroGuard Plus M4 (additional memory) price starts at $29, while the MicroGuard Plus Net prices starts at $55.

Unlike other "dongle" products, the MicroGuard Plus does not have any built in algorithms or developer ids. They are delivered in generic form. The software developer must enter the developer id, registered with MicroMacro, before they are shipped to the users. The developer also sets the seed and K-Code, used by MicroGuards one-way hash algorithm to verify the correct "dongle" is attached to the Macintosh.

Any applications written using the MicroGuard API automatically checks for a local "dongle", and then checks the Appletalk network for any MicroGuard servers. Thus, a single API handles both network and stand alone usage. The server, which is provide as part of the kit, is a face-less application and allows up to 50 users per MicroGuard Plus Network device attached.

MicroMacro posts their software development kit and manual on their web page for free download. They will send a free copy of the kit (software, manual and one "dongle") to any developers who request one. The kit includes Metrowerks C & C++ libraries, as well as the C & Pascal interfaces. There is a single, well documented C demo application. Also included in the kit is MicroMacro's utility program for configuring the device (MGDevTool). Finally, the kit includes various applications for running the Server version of the product.

The manual is very complete with in-depth explanations of uses for the device. The API is divide up into sections or "panels", and covers the different "dongle" functions a developer would expect. Since the entire device is configurable, different levels of password are used to insure that the correct access is allowed. When a portion of memory is to be Read Only in the developer application, the API only has the corresponding password to allow this. The "init" routines configure the device (enable and disable device, read and set ids, set addresses). The "password" panel changes any of the four different passwords. The "memory" routine accesses any of the volatile or non-volatile memory. The "calculation" routines handle the various K-Code functions to verify the device is the correct one for the software program. The "block" panel reads and writes to the various public and private blocks of memory, while the "bits" panel accesses bits of the bit array. The "counter" panel examines the built in counter registers.

MicroGuard also provides all the software and APIs needed to create one of the more powerful uses of a "dongle" device, an Identification & Authentication (I&A) server. Under this scheme, each client is connected to the server, which contains sensitive information. The clients are also connected to a local MicroGuard Plus device. Using the "dongle", the server can verify who is requesting the important information. An example of this might be a doctor client requesting patient information from a hospital server. This request can be over the Internet, or over a local area network.

FUTURES

All of the companies described here are working on new versions of their products. By the release of the iMac computer with the USB keyboard and mouse, developers will be able to order USB "dongles". These should work the same ways as the existing ADB based ones, and use the same API calls. USB has a much higher logical and physical limit of number of devices on the bus. Being able to "hot" plug in the USB devices will be a definite advantage. Also, the companies are working on various software only version of their projects. Along with, PACE <http://www.paceap.com/> and SoftLock <http://www.softlock.com/> new software only solutions for software licensing and distributing are being created. For stand alone and network Macintoshes, these software based, license management systems will not be as secure as the hardware ""dongles", but are becoming more popular because of cost and ease of distribution. Developers should look for announcements at MacWorld NYC, as well as keeping an eye on the Web pages.

THE DECISION

"Dongles" have many uses, both simple and complex. This article has attempted to explain many of these possibilities. If the reader is looking for a clear choice of "best dongle", they are on their own. For basic functionality, all three of the devices described here do exactly what they claim; they protect the software developers investment of time and money. The major features are almost identical. Each developer will have to compare costs, advanced features and ease of development. The best advice for the software developer is to pick the company he feels most comfortable working with. Ordering the equipment, modifying the existing code and handling the business aspects will be a partnership arrangement. Pick someone you can work with!

MT

IS YOUR SOFTWARE READY TO "RHAP"?

LICENSER KIT©

INTRODUCTORY PRICE $1995

- Create floating network licenses
- Tie license to host name
- Tie license to user name
- Tie to host ID
- Allow as many users as desired
- Have unlimited number of licenses on a network
- Add licenses on the fly
- Automatically unlicense if application is moved or copied
- Generate expiring licenses for "full strength" demos
- Works on **Yellow Box for Windows** as well as MACH/Rhapsody
- Comes with sample app and license key generator

FROM YOUR FRIENDS AT Caffeine Software & STONE DESIGN

www.stone.com/Licenser/ info@stone.com (505)-345-4800

by Matthew Xavier Mora
Edited by Peter N Lewis

Using the File Manager from MP Tasks

How to get data in and out of your MP Task

One of the most common complaints I received while supporting MP Library in Developer Technical Support was that you could not call the toolbox from an MP Task. Multiple preemptive tasks are not much use if you cannot get data into and out of them efficiently. This article shows one way to get data into and out of an MP Task using the file manager, however the techniques used here can be modified for other I/O operations (like audio, video or networking). "But, I thought you couldn't call the file manager from MP Tasks?" Well, you thought wrong. :-) Read on...

Background

In the early version of the MP library there was no easy way to call the toolbox because the MP Library was designed to be compatible with Copland's kernel tasking model. Since the Mac OS toolbox wasn't going to be available from Copland's kernel tasks, the same was done for the Mac OS version of the MP Library. After the Copland project was canceled it was decided to publish a few previously undocumented routines that let you work with the Mac OS toolbox from a task. One of the routines published is MPRPC. MPRPC is a remote procedure mechanism that lets you specify a routine to execute at a time when it is safe to make toolbox calls. It does this by suspending the task and then executing the supplied routine during SystemTask() time. The task is suspended until MPYield is called or until any toolbox routine calls SystemTask(). MPRPC is used internally in the MP Library to implement calls such as MPAllocate and MPAllocateSys (which is why these are blocking calls).

The code in this article is based on the MP File Library that I wrote before the MPRPC call was published. The MP File Library used MPQueues to communicate with the main task and have it execute toolbox commands.

Review

Let's review some of the MP programming guidelines and how adding blocking calls can change some of these guidelines.

1. Your tasks should do a considerable amount of work. If not, the benefits of using MP will be lost in the overhead of the scheduler and task switching. Adding blocking calls to your tasks adds additional overhead. The main benefit here is that by being able to call the toolbox from an MP Task your task can run autonomously from the main application thread. This results in a better user interface response from the application since the application can off load a time consuming task and call the main event loop more often giving the blocking calls more time to execute the toolbox calls.

2. You should allocate no more than (MPProcessors() – 1) number of tasks. While it is important to keep the number of tasks low so that task switching does not impact performance, adding blocking calls to a task will also hurt performance if nothing calls MPYield(). "Wait, I thought MP Tasks were preemptive?" Yes they

Matthew Xavier Mora was the engineer responsible for answering questions on Multitasking support in Apple's Developer Technical Support. As a self proclaimed evangelist for the Multi-processing API library he was instrumental in convincing both third-party developers and Apple engineers to implement MP support in their software. If you were ever thinking about moving into the Silicon Valley, consider that this article was written while Matt was sitting all night outside a school building waiting to register his son for pre-school. When Matt is not out doing crazy things like that you can reach him at mxmora@best.com.

are but if the task is blocked waiting on a resource, the resource can't be released until the main thread calls WaitNextEvent() or another task calls MPYield(). That being the case, if you use MPRPC calls it is a good idea to bend the n–1 rule and create an extra task that can help unblock any waiting tasks.

3. You should use MPQueues or MPSemaphores when communicating with MP Tasks. This does not change if you are using MPRPC so you should heed this warning.

Get on With It

OK, so how do I call the File Manager? For this simple example I will implement five MP calls that duplicate FSpOpenDF, FSClose, FSRead, FSWrite and SetFPos. Those are all the calls we need for a simple demo. We'll start with a FSRead type call.

First lets define a structure to pass to the MPRPC callback routines that will hold the values that we need to handle all the File Manager calls.

```
typedef struct FSParamRec{
   short       refNum;         // file ref num
   long        count;          // for read
   Ptr         buffPtr;        // for read
   FSSpecPtr   spec;           // for open
   short       permission;     // for open
   short       posMode;        // for setfpos
   long        posOff;         // for setfpos
   OSErr       result;         // error result
} FSParamRec,*FSParamRecPtr;
```

Now lets implement the callback routine that gets called at main application time. This routine will be executing at SystemTask time which means you can call any toolbox routine except for any routines that might call SystemTask() again.

```
static void * FSReadCallBack( void * parameter)
{
   FSParamRecPtr fsprp = (FSParamRecPtr)parameter;

   if (fsprp != nil) {
      fsprp->result = FSRead(fsprp->refNum,
                    &fsprp->count,
                    fsprp->buffPtr);
   }

   return fsprp;
}
```

First we check to make sure the parameter that was passed in is not nil then we simply call the File manager's FSRead call. When FSRead returns, we put the result into the result field and then return the pointer to the struct that was passed in.

All that is left is to do is to implement the new MyMPFSRead call.

```
pascal OSErr MyMPFSRead(short refNum,
                        long   * count,
                        void * buffPtr)
{
   FSParamRec fsrr;   // make the record on the stack
                      // no worries since it is a blocking call

   fsrr.refNum = refNum;
   fsrr.count    = *count;
   fsrr.buffPtr  = buffPtr;
   fsrr.result = paramErr; //preset in case
                           //anything goes wrong

   (void) _MPRPC(FSReadCallBack,&fsrr);
   //ignore what is returned


   *count = fsrr.count; //return the new count

   return fsrr.result;     //return the result
}
```

First we allocate a FSReadRec on the stack that gets passed to MPRPC. We fill out the fields in the struct with what was passed into us, call MPRPC and wait for the result. Then return the result to the caller.

That's it. You can now call FSRead from an MP Task. Using the same basic techniques you can implement all the file manager calls you need to get data in and out of your tasks. Now lets see how the task calls the new routines.

The MP Task itself is pretty straight forward as a result of the blocking I/O calls since there are no flags or spin loops to worry about.

```
static long MyMPTask(void * param)
{
   FSSpecPtr   fsp;
   Boolean        done = false;
   OSStatus    status;
   OSErr          err;
   MPQueueID      mpq = (MPQueueID) param;


   // don't start until we get the message
   status = MPWaitOnQueue(mpq,&fsp,nil,nil, kDurationForever);
   // the message is the file spec
   if (fsp) {
```

"IN JUST A FEW SECONDS I'VE SUCCESSFULLY TRACKED DOWN A MEMORY CORRUPTION BUG THAT HAS BEEN ELUDING ME FOR ABOUT 3 MONTHS. FANTASTIC :-)"
-BRYAN CHRISTIANSON-
SPOTLIGHT
The #1 Macintosh bug detection tool!

- Detect memory leaks automatically
- Instruction level bounds checking
- Validate 400 Mac Toolbox calls
- Pinpoint stale handle usage

- Integrated with all Mac debuggers
- Debug shlibs and stand alone code
- Faster than ever before

FREE DEMO

Finalist Macworld Editors' Choice

Onyx Technology

941.795.7801 Fax: 941.795.5901
www.onyx-tech.com sales@onyx-tech.com

```
    short       refNum;
    long        count = 1024; //read 1k of data

    err = MyMPFSpOpenDF(fsp,fsRdPerm,&refNum);
    if (!err) {
      err = MyMPSetFPos(refNum,fsFromStart,0);
      if (!err) {

#if qUseAsyncRead
          err = MyMPFSReadAsync(refNum,&count,gBuffer);
#else
          err = MyMPFSRead(refNum,&count,gBuffer);
#endif
        // we got some data. you could compress it
        // do FFT's on it or whatever.
        // In our case we just set the flag that we got
        // the data and tell the processors to sync up

        if (count > 0) {
          gCount = count; // signal that we got some text
          __eieio();      // sync processors
        } else {
          gCount = -1;    // signal that we got an error
          __eieio();      // sync processors
        }
      }
      err = MyMPFSClose(refNum);
    }
  }
}
```

In our task we immediately block (as every task should) on MPWaitOnQueue waiting for the FSSpecPtr from the application. When MPWaitOnQueue returns, we check the file spec pointer to make sure it is not nil and precedes to open the file. We set the file position to the beginning of the file and start the read operation. Notice that for either the async or non async case the code is still the same. The only difference is to the application since the task is blocked until the read completes. After the read completes, this is where you would do some serious processing on the data. It is very important that you do a lot of processing to minimize the overhead of the blocking I/O calls. The demo doesn't do any processing so the next thing to do is to set the gCount variable indicating we got the data making sure the write get synchronized with the other processors. We close the file and return. Returning from the task kills the task. You might want the task to hang around and be ready to process another file. In that case set up a while loop on MPWaitOnQueue. You can set the exit termination condition to be a nil FSSpecPtr.

ADDING MORE FEATURES

The FSRead technique is good at getting data in and out of your task but you basically block the entire application while it waits for the FSRead to complete. We can improve this by using asynchronous file manager calls to keep from blocking the main application task while executing a Read call.

We need a different structure to do an async read. I wrap the new struct around a ParamBlockRec to contain the flag needed to signal the completion of the read call.

```
typedef struct FSReadAsyncRec {
  ParamBlockRec    pb;              // standard paramblock
  Boolean          callPending;     // our pending flag
} FSReadAsyncRec, *FSReadAsyncRecPtr;
```

The MyMPFSReadAsync code is a little more complicated but it saves having to have another task running just to call MPYield() since this routine spins on MPYield waiting for the PBRead to complete.

```
static pascal OSErr MyMPFSReadAsync(short refNum,
                                    long  * count,
                                    void * buffPtr)
{
  FSReadAsyncRec     fsrar;    // make the rec on the stack
  // Build a rountine descriptor by hand since we can't call
  // NewIOCompletionProc(userRoutine)
  RoutineDescriptor ioCompProc =
            BUILD_ROUTINE_DESCRIPTOR(uppIOCompletionProcInfo,
                                          MyReadCompletion);

  ClearBlock(&fsrar,sizeof(fsrar));

  fsrar.pb.ioParam.ioRefNum      = refNum;
  fsrar.pb.ioParam.ioReqCount    = *count;
  fsrar.pb.ioParam.ioBuffer      = buffPtr;
  fsrar.pb.ioParam.ioCompletion  = &ioCompProc;
  fsrar.callPending              = true;
  __eieio();                   //ensure that callPending gets set
                               //before we call MPRPC

  (void) _MPRPC(FSReadAsyncCallBack,&fsrar); //ignore what is

  // spin waiting for flag to be set in completionRoutine

  while ( fsrar.callPending ) { //Spin waiting for completion
    MPYield();
  }

  *count = fsrar.pb.ioParam.ioActCount;
            //return the new count

  return fsrar.pb.ioParam.ioResult; //return the result
}
```

MyMPFSReadAsync sets up the parameter block, builds a completion routine descriptor on the fly, calls MPRPC and then spins in a tight loop calling MPYield until the callPending flag is cleared.

The FSReadAsyncCallBack routine is very simple.

```
static void * FSReadAsyncCallBack( void * parameter)
{
  FSReadAsyncRecPtr fsr = (FSReadAsyncRecPtr) parameter;
  OSErr err;

  if (fsr != nil) {
    err = PBReadAsync((ParmBlkPtr)fsr);
                               //just call PBRead and return
  }                            // completion routine sets the flag
  return fsr;
}
```

FSReadAsyncCallBack just calls PBReadAsync and returns. Below is the completion routine that tells the task the read has completed.

```
static void MyReadCompletion(ParmBlkPtr pb)
{
  FSReadAsyncRecPtr fs = (FSReadAsyncRecPtr)pb;

  fs->callPending = false; // set flag
  __eieio();                 // make sure it sticks
}
```

It just sets the callPending flag, signals the processors to sync up and returns. We can't set a MPQueue or a MPSemapore in here (which would be the better way to do it) because MP Library calls can't be called at interrupt time.

Handling asynchronous routines gets a little more complicated but it saves having to make sure other tasks are running just to call MPYield(). Now you might be thinking why are we using a flag when you could just spin on ioResult? Read on to see why this is not good idea...

Gotchas

When working with multiple processors some conventional Mac programming wisdom goes out the window. A good case in point is when ioResult is set. Normally ioResult is set to 1 to indicate a call is pending. The last thing the file manager does before calling the ioCompletion routine is to set ioResult to the error result from the parameter block call. None of this really changes when multiple processors are involved but the non-main processors are not bound by the 68k enable/disable interrupt tricks. So if your MP Task spins on ioResult waiting to see when the read is complete (ioResult != 1) your task starts to execute before the file manager is done with the parameter block. After the file manager sets the ioResult field, it gets the ioCompletion routine's address from the parameter block and jumps to it.

In our case the parameter block in on the stack and when the task unblocks, the stack is released and your task crunches merrily along where a parameter block used to be (and is still in use by the file manager). The second processor could be a 200 MHz CPU and in the time the file manager has set ioResult and jumps to the completion routine, your task could be millions of instructions away using the memory where the parameter block used to be.

The same is true for many of parts of the Mac OS Toolbox. The critical region technique of disabling interrupts does not work well when multiple processors are involved. So be careful and always use MPQueues, MPSemaphores and MPCriticalRegions to coordinate your various tasks.

Another gotcha may be in your thought process. You might be thinking that it would be cool to use the same techniques mentioned in the article to make every Toolbox call available from MP tasks. While this is possible, and would make your task code a lot easier to write, it is not a good idea. The benefits of multiprocessing only come from careful algorithm design, implementation, and profiling. Guideline #1 mentioned above says that your task should do a considerable amount of work to gain any performance improvements. Having your task block, waiting on a bunch of toolbox calls is not going to improve performance. On the other hand having to load all the data you need into memory before your task can start running may not be feasible either. This is where a careful balance of having main processor moving data in and out of your task while processors n+1 crunch along can really pay off.

More MP Information

Hopefully, this article piqued your interest in Multiprocessing. If you want more information there are a number of documents and resources to help you get the most out of MP. An introduction to MP systems was printed in MacTech March '96, TechNote 1071 on Multiprocessing is on the web <http://www.apple.com/developer/> and I have set up a MP mailing list where developers can ask questions on MP programming issues. The list includes folks like the senior engineer who wrote the MP Library as well as Chris Cooksey and myself. For subscription information you can go to my web site <http://www.best.com/~mxmora/mxm.html>. Also, don't forget Apple Developer Technical Support is there for information about MP's past, present and future.

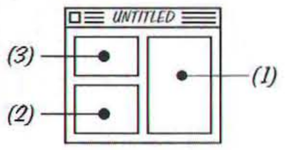
Summary

I hope this article shows how easy it is to get data into and out of your MP tasks. Use this information wisely and you should see some real improvements in your applications performance. You can use these techniques to work with other I/O technologies like networking, graphics and sound. I have created a MP File Library that you may want to use based on some of the techniques used in this article. It uses a slightly more complicated model for better performance. You can download a copy of my MP File Library from my web site at <http://www.best.com/~mxmora/software.html>. Good luck, and happy multiprocessing.

MT

Quadrivio General Edit

Debug & Analyze Complex File Formats.



1) Define your format with familiar, C-like statements.
2) View, analyze, and edit data file in up to 5 columns.
3) View and edit file parameters. Also edit handle blocks and other data structures in memory.

Free!
Lite & demo versions available on web site.

✔ **Quick insight** into data structure contents.
✔ **Easier** than studying MPW's dumpfile output.
✔ **Faster debugging** with clear display of data.
✔ **Saves coding time** with direct data editing.

Visit our web site for immediate download.
On-line ($195) and boxed ($249) versions available.
Also available from SciTech International 1-800-622-3345.

http://www.quadrivio.com

Quadrivio Corporation info@quadrivio.com
1563 Solano Avenue #360 Berkeley, CA 94707 (510)524-3246

by Jeff Clites <online@mactech.com>

Danger, Will Robinson

When I sit down to program, I always know, somewhere in the back of my mind, that my efforts are ultimately targeted to the screen, the disk, or (if I'm feeling adventuresome), the network. But in the end, it's a very passive endeavor. My programs produce information — maybe images or sound — but, not action; they don't make things move. But they can. This month, we are going to find out about hooking your Mac up to the outside world, in a more visceral way than just attaching a printer. It's a hands-on subject, so roll up your sleeves and get ready.

Make Your Own Friends

Robots. They're as close as you can get to a computer with legs, and they can have almost as much personality as a Macintosh. I first saw a fun robot called CHiP at Macworld SF in January. It's a little blue car with two wheels and two sensing bumpers, which you can program and then run either tethered to your Mac or autonomously. The Hyperbot web site has a QuickTime movie of CHiP navigating a maze, to give you an idea of what he is like. There are also facilities for attaching additional sensors to CHiP, so for instance you could attach a light sensor and have him hide in the shadows. CHiP is still under development at the time of writing, but when finished it should be available through BeeHive Technologies, the makers of ADB I/O, reviewed elsewhere in this issue.

Hyperbot
<http://www.hyperbot.com/>
BeeHive Technologies, Inc.
<http://www.bzzzzzz.com/>

Another fun place to get your feet wet is with LEGO® robots, built from LEGOs® augmented with a few electronic parts. The Caltech ICOBotics™ site has Macintosh software which lets you write programs for your robots, using an iconic language. ICOBotics™ is intended to be an educational tool, and it would be a great way to introduce kids to programming, letting them learn the fundamentals (control structures, subroutines, etc.) while seeing very tangible results.

ICOBotics™ for LEGO Dacta® Home Page
<http://www.micro.caltech.edu/icobotics/>

There are also a number of pages on the web devoted to those seeking a more hands-on approach. Some of them are specifically geared (no pun intended) to the hobbyist roboteur, but the same general principles apply to building any sort of device which moves — so even if you don't want to build R2-D2 a brother, you still might have fun adding computer control to your doggie door. A good place to start is Dan O'Sullivan's Physical Computing page, a hand-illustrated guide to the various components involved in hooking a computer up to the outside world, focussing on communication through the serial port and targeted toward artists. Matthew McDonald has a large compendium of useful information on hobby robotics, and there is a well-organized FAQ for the comp.robotics.misc and comp.robotics.research newsgroups. For the latest news, stop by Robotic.com, which prominently displays a "made with Mac" logo.

Physical Computing
<http://www.itp.tsoa.nyu.edu/~alumni/dano/physical/physical.html>
Info on Hobby Robots
<http://www.cs.uwa.edu.au/~mafm/robot/index.html>
Robotics FAQ
<http://www.frc.ri.cmu.edu/robotics-faq/TOC.html>
ROBOTIC.COM
<http://www.robotic.com/>

For further technical information as well as supplies, start with FerretTronics. They sell controllers for servo motors, and their web site has a step-by-step guide to using their package to build the circuitry of a robot or other device, including Mac-specific information. They also have free Java-based Mac software for interfacing with their controllers, and examples showing how to drive them from AppleScript or Future BASIC.

FerretTronics
<http://www.busprod.com/ferrettronics/>

For programming of embedded microcontrollers, Crossbow by Peripheral Issues is a Macintosh cross-assembler and IDE which allows you to target a wide range of chips, and Interactive C by Newton Research Labs is a programming environment created for the MIT LEGO® Robot Design contest, and it can compile instructions on-the-fly for quick development. You will also want to look into the Handy Board, developed at MIT as well. It's a controller board for experimental mobile robotics, and should be programmable in either of the above environments. By the way, it was designed on a Macintosh (don't forget to tell your PC buddies), and they provide links to the circuit-design software used.

Crossbow
<http://www.periph.com/crossboweb/Crossbow.html>
Interactive C
<http://www.newtonlabs.com/ic/>
The Handy Board
<http://lcs.www.media.mit.edu/groups/el/projects/handy-board/>

For the rest of the parts you might need, the Internet Robotics Sources page has an extensive list of commercial (and other) links, and the Mondo-tronics' Robot Store claims to have "the world's biggest collection of miniature robots and supplies," and is worth visiting just to see their cute web design.

Internet Robotics Sources
<http://129.79.254.195/l/www/robotics/world.html>
Mondo-tronics' Robot Store
<http://www.robotstore.com/>

Now, when you tell you friends, "my Mac can beat up your Mac," you'll really mean it.

Respect for the Elderly

If you are anything like me, you hate to give up a good Mac which has served you well, even after you've purchased a new one. So you've saved Old Faithful, but what now? Fortunately, there's a lot you can do with an old Mac — from the silly (yes, we've all heard of the Macquarium) to the practical. Jeff Weitzman's Great Ideas for Old Macs page has a bunch of suggestions, such as using old Macs to set up email and ftp servers for an intranet, or using them to run a book tracking system in a school library. To find software for your early Macs, go to Jag's world, which has an extensive list of shareware which runs on older Macs, and even has instructions for "How to Get Your Compact Mac on the Web." Finally, there's the Low-End Mac page, which has all sorts of information on older Macs and also tracks current Mac-related news.

Great Ideas For Old Macs
<http://weitzman.net/classicmacs/>
Jag's World
<http://www.eden.com/~arena/jagshouse/jagshouseone.html>
How To Get Your Compact Mac On the Web
<http://www.eden.com/~arena/jagshouse/GetYourCompactMacOnTheWeb.html>
Low End Mac
<http://come.to/lowendmac>

Just Because

While we are on the subject of the fun and unusual, I have to point out the Oddities Curios & Rarities for Macintosh web site. It has nothing to do with programming per se, except to drive home the point that necessity is not the only mother of invention. It is full of all sorts of downright weird software — things such as Sim-Tofu, Psychomatic, and Kant Generator Pro. It all makes you wonder what you would do with that much free time.

Oddities Curios & Rarities for Macintosh
<http://www.mac-curios.com/>

These and scads of other links are available from the MacTech Online web pages at <http://www.mactech.com/online/>.

MT

HELP MAKE MACTECH WORK

Here at *MacTech Magazine*, we rely heavily on outside writers for most of the material that appears in our pages. If readers did not participate in the magazine, sending us their ideas and taking the time to write articles, there would be no *MacTech*. *MacTech Magazine* is not a staff of writers sending a constant stream of one-way messages outwards; it's a living, evolving network of readers conversing with one another, educating one another, sharing their knowledge, their experience, their interest, their trials and tribulations and joys and successes in the constantly unfolding story of programming the Macintosh. *MacTech Magazine* doesn't just happen: it's what the community makes it. If we carry reports of future trends and technologies, if we teach useful methods, if we review new books and tools, if we provoke thought, provide help, ride the wave of current interests and concerns, it is only because we reflect the thoughts of our readers, who speak through our pages.

You are invited to involve yourself in this exciting conversation amongst readers. No matter who you are, no matter what your credentials may be, if you have a tale to tell, a trick to share, a technique to teach, we want you to consider joining the family of those who write for *MacTech*.

Don't just wait for a topic to be covered or a technique explained in *MacTech!* Take responsibility! Write us an article yourself!

To write for *MacTech*, just send for our Writer's Kit. It's a Microsoft Word file containing the Styles you need to use, and giving lots of helpful advice and information, including all the legal stuff. You can let us know what you're writing about, or, if you want to, you can just write the article and spring it on us when it's done. [**Note:** We also have a need for people willing to make themselves available to write occasional product/book reviews.] If we publish your article, you'll be paid for it!

Write to us, the editorial staff, at editorial@mactech.com (or one of the other addresses listed on page 2 of the magazine). Take the future of *MacTech Magazine* into your own hands!

For Macintosh Programmers & Developers
MacTech®
MAGAZINE

by Jessica Courtney <managing_ed@mactech.com>

New Products and Silicon Alley Attendees at Macworld Expo Demonstrate Mac-Market's High Energy and Innovation

Attendees invigorated by Steve Jobs' In-person Keynote, iMac showcase and Apple's Announcement of Third Consecutive Quarterly Profits

Over 34,000 consumers, educators, creative professionals, and sales and support staff jammed New York's Jacob K. Javits Center for IDG's Macworld Expo '98 last week. Attendees to this year's East Coast show tested an array of innovative, new Mac-based products, heard from the best-known names in educational technology and digital artistry, and participated in the first-ever gaming contest.

Throughout the three-day show, held July 8-10 at the Jacob K. Javits Center, a number of welcomed announcements were made. Attendees clearly rallied around Apple Computer, Inc. when interim CEO Steve Jobs announced during the keynote — in person rather than via satellite — that the computer maker expected to post net profits for the third straight quarter. This week, Apple announced that the last quarter was its best in three years, with reported net income of $101 million. For the last two quarters, revenues have stabilizing at about $1.4 billion. Keynote Highlights Jobs joined by Phil Schiller, Apple's Vice President of worldwide product marketing also announced that Apple's new $1,299 iMac personal computer would hit the stores August 15 installed with 56K modems rather than 36.6K ones, as originally planned. Jobs also told the enthralled Macworld Expo audience that 177 new Macintosh programs had been announced in the 63 days leading up to the conference a clear indication of market support. He bolstered the good news from Apple with a live demonstration of the iMac computer, Apple's latest entrant into the competitive personal computer realm.

Significant announcements from Microsoft and Disney OnLine executives further reinforced Job's positive predictions for Apple's future.

Consumer Appeal

Apple Computer featured its new iMac and its complete line of PowerBook G3 and Power Macintosh G3 computers, as well as QuickTimeŒ 3 and WebObjects software products.

Gaming developers, major entertainment companies, broadcasters, personal finance and productivity application developers and hardware manufacturers provided consumers hands-on experience with their latest products in a new "Content @ Home" pavilion.

Additionally, the competitive types participated in the first-ever National Macintosh Gaming Championship, which highlighted the latest games for the Macintosh platform.

Education Appeal

The Education District Building on its education community support, Apple Computer sponsored the Education District pavilion, which featured software and Internet tools, curriculum building solutions and multimedia for schools, training and home learning.

Professional Appeal

The Macworld/Pro Conference Professionals attending this year's Macworld Expo — the Creative World, participated in the highly regarded Macworld/Pro conference, Macworld Users conference and day-long pre-conference workshops.

Attendance Demographics

According to an on-site survey of Macworld Expo attendees: more than 96 percent came to Macworld Expo to see the new products; 87 percent to keep up-to-date on industry trends; and, 68 percent to compare products with current vendors. Nearly 50 percent of all attendees were their organization's final product-purchasing decision makers. Seventy-four percent said they believed the event is important in determining their upcoming computer and peripherals, and software purchases. Seventy-eight percent said the show was "well worth their time and money" and 82 percent would recommend Macworld Expo to a colleague.

Exhibitor and Attendee Figures

More than 230 exhibitors set up shop at Macworld Expo. The number of attendees at this year's Macworld Expo — New York was 34,200. Thirty percent of the attendees worked in the creative industry, 25 percent were from the education community, and 20 percent were from the reseller, value added service and support community representing a significant industry interest in supporting the latest Mac products. In addition to the business audience almost 20 percent of the attendees were general consumers drawn to see Apple's new iMac computer and the new range of iMac-related products.

<http://www.macworldexpo.com>
<http://www.comnetexpo.com>
<http://www.idgexpos.com>

MT

by Steve Sisak <tips@mactech.com>

Gotcha — Calling GetResource()

If you call Get1Resource(), expecting to get a resource from the topmost resource file, you can be unpleasantly surprised. With the introduction of the "Fonts" folder in System 7.1, a patch was made to the resource manager so that all the files in the fonts folder will appear to be part of the System file, even though they are not.

Most of the time this is not a problem, since most people don't care if a resource is in the System file or in a file in the fonts folder (or in the enabler, or in ROM).

However, if you absolutely need to know the file where a resource is loading from, you can use HomeResFile(). It will return the refNum of the resource file the resource came from.

Marshall Clow, Adobe Systems

Determining if a Resource File was Actually Opened

When you use any of the Resource Manager calls for opening a resource file, and the resource file is already open by the Resource Manager with that access, it will not open the file again. Instead, it returns the refNum of the already-open reference, and also does the equivalent of a UseResFile() on that refNum.

This means you have to be careful if your code is capable of opening and closing arbitrary resource files — if the Resource Manager didn't actually open the file for you (for example if you were to open your own application, or the System file, or a suitcase in the Fonts folder), then you must not close it with CloseResFile().

So how do you tell if the resource file was actually opened for you or not?

It's easy: if the resource file was opened, then its resource map will be added to the top of the in-memory chain of resource maps (pointed to by the "TopMapHndl" low-memory global). If not, then the chain remains unchanged.

The following sequence illustrates the idea:

```
SInt16    TheResFile, PreviousResFile;
BooleanReallyOpened;
Handle    PreviousTop;

PreviousResFile = CurResFile();
PreviousTop = LMGetTopMapHndl();
TheResFile = FSpOpenResFile(ResFileSpec, Permissions);
```

/* checking for errors omitted */

```
ReallyOpened = LMGetTopMapHndl() != PreviousTop;
```

/* do your processing on the resources, then when finished, restore things as they were: */

```
if (ReallyOpened)
    CloseResFile(TheResFile);
UseResFile(PreviousResFile);
```

Lawrence D'Oliveiro, Hamilton, New Zealand

Serial Port Tidbits Comments

Thanks for the informative article bringing together a lot of previously hard to find information. I spent many hours digging around (and cursing!) trying to find that same info.

I had written a similar routine to your FindPortInfo() routine, and after checking mine, I initially thought there might be a bug in yours. I finally realized that your code was in fact right, but there is a subtlety which might not be apparent to readers.

In the line:

```
while ((crm = CRMSearch(crm)) != nil)
```

a key point is that you are reassigning to crm the result of the search.

In my code I was using a temporary variable to pass to CRMSearch(), as in

```
while ((foundCRM = CRMSearch(tempCRM)) != nil)
```

This means that in my code, after the EqualString() check, I need to insert:

```
tempCRM->crmDeviceID = foundCRM->crmDeviceID;
```

to avoid a potential infinite loop.

Your code (and the code in the CTB book) avoids this by using the same variable in the assignment, but that also hides this subtle point CRMSearch() starts looking from the device ID passed to it in the crmDeviceID field of the CRMRecPtr passed to it.

One other thing I'd like to mention is that instead of your IsDriverOpen() routine, you could call the Connection Resource Manager routine CRMIsDriverOpen(), described in the Communications Toolbox 1.1 Engineering Notes (not in the original CTB book — the 1.1 notes are another hard-to-find source...)

Jim Wintermyre, E-mu Systems, Inc.

Ed Note: The fact that CRMSearch() looks at the deviceID and deviceType passed in and returns the device of the same type with the next higher ID. This is a subtlety worth noting. Also, the CTB 1.1 Engineering Notes are an important resource I forgot to mention in the original article.

—sgs

by R. D. Warner
Edited by Michael Rutman

Bitmap Graphics

Demonstrated in Ray Tracing Algorithm Code Snippet

Overview — Why Use Bitmaps?

There are times when the programmer needs control over individual pixels, plus all the speed that is possible. Rendering in 3D is one case where individual pixel values are calculated using compute-intensive algorithms. Lighting calculations are performed for each pixel (often recursively) and every little coding trick eventually is used in the pursuit of speed.

Pswraps is one technique available to Rhapsody developers. The "wraps" are C function calls that perform one to many PostScript operations in one interprocess message to the Window Server. So it is an efficient technique, and it is possible to write to individual pixels...sort of. For instance, one can define a rectangle with area of one pixel and the appropriate color, and draw that. This means one function call to draw each pixel. Display postscript is not really geared to individual pixel manipulation.

A slightly lower level approach is to use bitmaps. A data buffer is populated dependent on various color schemes (monochrome, RGB, CYMK, HSB) and other factors, such as whether or not the alpha channel is used. Then that data is rendered all at one time to whatever NSView has focus locked. One can expect a boost in performance using this approach and indeed that is the case. The pswraps code is simpler to use, but slower.

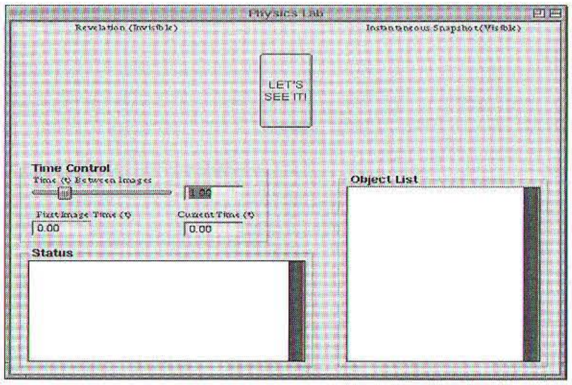
From games to medical imaging to paint programs, the need for high-speed pixel manipulations is the driving force in the decision to use bitmaps. This article uses a simple 3D rendering application as test code for comparing performance of a pswraps implementation against a bitmap.

Included in this article is a code snippet from *Physics Lab.* It is ported to the Rhapsody environment but this code should basically work fine in OPENSTEP 4.2 also. This program uses a ray tracer algorithm to *render* a typical scene composed of various graphics primitives (such as spheres, planes, cylinders, etc.) into an NSView object. There are many algorithms for achieving shading effects on 3D objects and ray tracing is one of the most popular ones for achieving photorealistic lighting effects. Rendering is the term used for the process of generating the 3D scene based on mathematical models. The NSView object is one of several in the Rhapsody AppKit that can display an image on the screen.

The application also includes a second NSView alongside the first where a custom renderer such as a subclassed ray tracer, can be displayed (disabled in screen shots). This is useful in visualization projects where one wants to see how something would be seen "normally" plus how it would appear using the custom viz algorithms. Specifically, this will be used for the visualization of field phenomena such as gravity — one view displays the collection of objects as they would normally be seen with visible light, and the other view will make a visible representation of the field interactions between and inside the objects. (See **Figures 1-3**)

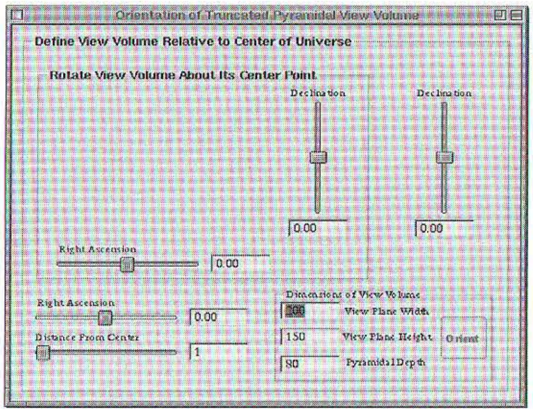
One will learn something about NSImage classes in this article too. They support bitmaps plus many other types of image representations. In fact it is necessary to understand them before advancing further.

Richard Warner is an independent consultant living in Colorado. He worked for over eight years with the USDA as first a computer specialist then a computer scientist. His small company, Perceptual Research Ventures, has published several technical articles in Radio-Electronics magazine, plus designed and marketed the Synergy Card expansion card for PCs and compatibles. Current projects include the Physics Lab viz application. He may be contacted at: rwarner@prv.com.



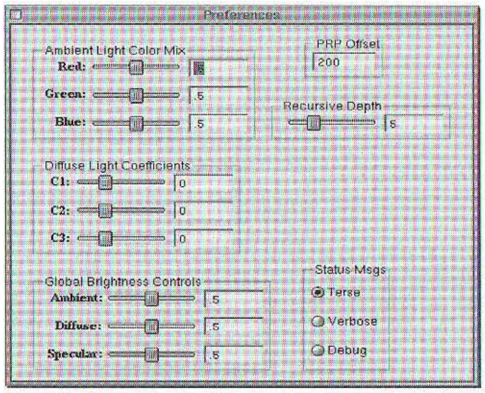
Time (s) Between Images: 1.00
First Image Time (s): 0.00
Current Time (s): 0.00

Figure 1. *Main window in Physics Lab with second NSView disabled.*


Orientation of Truncated Pyramidal View Volume
Define View Volume Relative to Center of Universe
Rotate View Volume About Its Center Point
Orient

Declination: 0.00
Declination: 0.00
Right Ascension: 0.00
Right Ascension: 0.00
Distance From Center: 1
Dimensions of View Volume
200 View Plane Width
150 View Plane Height
80 Pyramidal Depth

Figure 2. *Window for orienting the view volume.*



Ambient Light Color Mix
Red: .5
Green: .5
Blue: .5
PRP Offset: 200
Recursive Depth: 5
Diffuse Light Coefficients
C1: 0
C2: 0
C3: 0
Global Brightness Controls
Ambient: .5
Diffuse: .5
Specular: .5
Status Msgs
[✓ Terse]
Verbose
Debug

Figure 3. *Preferences Panel.*

NSImage Classes

Description

An NSImage is a lightweight object compared to an NSView. An NSImage can draw directly to the screen, or be used (as it is in this code) as a container for NSImageRep subclass objects. The container composites the image to an NSView...which draws to the screen. There are a number of configurations for using the various view-related objects in AppKit, so this is only one approach. This approach is especially useful if one has multiple NSImage objects drawing to different areas within an NSView. Perhaps a collage-type scene where individual images (each one represented by an NSImage) over time are moved around within the rectangle defined by the NSView.

The NSImage contains an NSMutableArray object that can contain numerous NSImageRep subclass objects. NSMutableArray replaces the NXList found in NEXTSTEP 3.3. It is a dynamic array for containing groups of objects. Each representation is for the same image. For example, one can have representations that are at various resolution levels, or using different formats such as TIFF and EPS. Different color schemes can be used as mentioned earlier and one can even make custom representations to render images from other types of source information.

A representation can be initialized in a variety of ways, somewhat dependent on the representation type. The most straightforward ways are to initialize the NSImage from a file (typically in the application bundle) or from an NSData/NSMutableData object created dynamically, and let it create the appropriate representations (which it manages), based on the header information in the data. The image can also come from the pasteboard, from an existing NSView, or raw bitmap data. Reading in a file of an unsupported format type requires either a custom representation or a user-installed filter service.

The real magic of NSImage classes comes from its ability to select the best image for the current display device, from its list of representations. This logic algorithm is somewhat customizable by the programmer. There are several flags that can be set to modify the selection criteria and priorities (see below). The algorithm naturally only can select from the NSImageReps that are managed by the NSImage in question. Thus the selection process can also be controlled indirectly by the types of representations added. The steps in the selection algorithm below are followed only until the first match is established.

Selection Algorithm

1) Choose a color representation for a color device, and gray-scale for a monochrome device.

2) Choose a representation with a matching resolution or if there is no match, then choose the one with the highest resolution. Note that setPrefersColorMatch:NO will cause the NSImage to try a resolution match before a color match.

3) Resolutions that are multiples of the device depth are considered matches by default. Choose the one closest to the device. Note that setMatchesOnMultipleResolution:NO will cause only exact matches to be considered.

4) The resolution matching discriminates against EPS representations since they do not have defined resolutions. The setUsesEPSOnResolutionMismatch:YES will cause the NSImage to select an EPS representation (if one exists) if no exact resolution match can be found.

Trying to BREAK into programming?

Visit Developer Depot!

Developer Depot has the latest tools and reference materials to help you get started in programming.

At any time of the day or night, the Depot online store provides access to the latest tools for your development projects — with guaranteed satisfaction.

<http://www.devdepot.com/>

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200
Voice: 800/MACDEV-1 (800/622-3381) • Outside US/Canada: 805/494-9797
Fax: 805/494-9798 • E-mail: orders@devdepot.com

5) Choose the representation with the specified bits per sample that matches the depth of the device. Note that the number of samples per pixel may be different from the device (an RGB no-alpha color representation has three samples per pixel, while a monochrome monitor would have one sample per pixel, for example).

6) Choose the representation with the highest bits per sample.

Abbreviated Method Quick Reference (based on online documentation)

```
-(id)initWithContentsOfFile:(NSString*)filename
```

Initializes the NSImage with the contents of filename and reads it at this time. If it successfully creates one or more image representations, it returns self. Otherwise the receiver is released and nil is returned.

```
-(id)initWithData:(NSData*)data
```

Initializes the NSImage with the contents of data. If it successfully creates one or more image representations, it returns self. Otherwise the receiver is released and nil is returned.

```
-(void)addRepresentation:(NSImageRep*)imageRep
```

Adds imageRep to receiver's list of managed representations. Any representation added in this way is retained by the NSImage, and released when it is removed.

```
-(void)removeRepresentation:(NSImageRep*)imageRep
```

Removes and releases imageRep.

```
-(NSArray*)representations
```

Returns an array of all the representations managed by the NSImage.

```
-(void)lockFocus
-(void)lockFocusOnRepresentation:(NSImageRep*)imageRep
-(void)unlockFocus
```

Using the approach outlined in this code one does not have to bother with lockFocus/unlockFocus pairs. This is because the NSImage performs its drawing to the NSView from within the NSView's own drawRect: method. That method is called indirectly by the NSView when it receives a display message from within the code. Focus is automagically locked on the NSView as part of this whole process, so it does not need to be done by the programmer.

These methods are mentioned here because one *will* use them in certain circumstances. If the NSImage draws directly to the screen instead of to an NSView, these may be needed. To force the NSImage to determine its best representation, or to test in advance that an NSImage can actually interpret its image data, one may need to lock the focus. In the last case, if the NSImage cannot understand the data it was given, it will *raise an exception* when it fails to lock focus; the exception can be trapped by the code to determine that the image file was the wrong format, garbled, or in general not displayable.

Exceptions are a programming technique within the OPENSTEP/Rhapsody environment that effectively take the place of error return codes and code for checking the return codes. See the documentation for NSException objects for more details.

The act of locking focus does a variety of things but conceptually it determines what object will be rendering the image information to the screen, and it establishes the coordinate system that will be used to position the images.

```
-(void)compositeToPoint:(NSPoint*)aPoint
    operation:(NSCompositingOperation)op
```

The aPoint argument specifies the lower-left corner in the NSView coordinate system as to where the NSImage shall be composited. The op argument can take one of currently fifteen enumerated values. The NSCompositeCopy is common if one simply wants to copy the NSImage image into the NSView. If the alpha channel is used, NSCompositeSourceOver would be used (source laid over destination with a specified opacity). NSCompositeClear will clear or erase the affected destination area. NSCompositeXOR does a logical exclusive-OR operation between the source and destination data.

```
-(void)dissolveToPoint:(NSPoint*)aPoint
    fraction:(float)aFloat
```

Composites the image to the location specified by aPoint, but it uses the dissolve operator. The aFloat argument ranges between 0.0 and 1.0 and indicates how much of the resulting composite will be derived from the NSImage. A slow dissolve can be accomplished by repeatedly using this method with an ever-increasing fraction until it reaches 1.0. Note that each dissolve operation must be performed between the source data and the *original* destination data, not the cumulative result of each dissolve (keep a copy of the original data somewhere and do the dissolve operation off-screen — then flush it to the screen). Note that a slow dissolve is not possible on a printer, but one can do a single dissolve operation and print the results of that.

```
-(void)setFlipped:(BOOL)flag
-(BOOL)isFlipped
```

When working with raw bitmap data one may find it necessary to flip the coordinate system y-axis. Thus (0,0) is in the upper-left corner instead of the lower-left corner. It depends how the algorithm generates the data, but if images are being displayed on the screen upside down, this is how to fix it:

```
setFlipped:YES
-(void)setSize:(NSSize)aSize
-(NSSize)size
```

With raw data in particular, one wants to explicitly make the image representation (since the structure of the data needs to be described), then add the representation to the list of managed representations for the NSImage. Contrast this approach to the situation where one initializes an NSImage with the contents from a file, and the appropriate representation(s) are created automatically. When working with raw bitmap data the image representation must manually be created and told how many bytes per pixel there are, whether or not there is an alpha channel, etc. Since the NSImage did not create the image representation it does not know the size of the image. Use setSize: to tell it.

```
-(NSData*)TIFFRepresentation
```

Returns a data object with the data in TIFF format for all representations, using their default compressions.

Example Uses of an NSImage

1) Draw same image repeatedly to an NSView (offscreen cache).
2) Have optimized images for various types of displays, or printer.
3) Read file in one format and write to another (e.g., read bitmap, write TIFF).
4) Draw to a bounded rectangle within an NSView.
5) Manipulate individual pixels.
6) Read image from an NSView, perform filtering operation, draw image back to an NSView.
7) Draw to an NSView using a variety of logical operators other than simple copy.
8) Make existing image in NSView dissolve into new image.

NSBitmapImageRep Classes

Description

There are currently four subclasses of the NSImageRep class: NSBitmapImageRep, NSCachedImageRep, NSCustomImageRep, and NSEPSImageRep. For manipulating raw bitmap data, use the bitmap image rep. The class has a data buffer of type: unsigned char*. There are numerous initializers, but for raw bitmap data such as that which will be generated in this example, use the initWithBitmapDataPlanes::::::::: method. All the many arguments are used to describe the structure of the data.

It takes a lot of arguments to canonically describe raw bitmap data. For example, there is a "meshed" mode where the data color components are grouped together in memory. An RGB pixel would typically have three bytes together, one for each color component—possibly a fourth byte if the alpha channel is used. There is another mode called "planar" where each color component is grouped in a separate plane. This means in the previous example all the red bytes would come first, then the green, then the blue, then the alpha.

Other arguments define the width and height of the image in pixels, plus the number of bits per pixel and the total number of bytes in the image. There are instances where the total number of bytes is different from the number of pixels times the number of bits per pixel such as when the pixels are aligned on word boundaries and the pixel bits are less than that. So it may seem to the programmer that more arguments are being provided than is necessary, but this is not the case. The programmer can also specify which color space is used and whether or not the alpha channel is present.

Currently, no less than nine color spaces are supported, including both calibrated (device independent) and device-specific versions. The code snippet in this article uses the RGB color space with no alpha, one byte per color component, and three bytes per pixel.

One of the arguments to the initializers is a pointer to a data buffer. This may be set to NULL. If so, the method will calculate the size of the buffer (based on the arguments given it) and allocate it. The buffer is thus owned by the instance, and freed when it is freed. The getBitmapDataPlanes or bitmapData methods are used as accessors to get a pointer to the buffer that was allocated, so that it can be populated. The other approach is to allocate the memory for the buffer before initializing the representation. If this is the case then one does not have to use the data method to retrieve a pointer since one already knows it. The instance does not own the buffer and it is the programmer's responsibility to explicitly free it.

Abbreviated Method Quick Reference (based on online documentation)

```
+(NSArray*)imageRepsWithData:(NSData*)bitmapData
```

Creates and returns an array of initialized NSBitmapImageRep objects based on images in the bitmapData. Returns empty array if images cannot be interpreted.

```
-(id)initWithBitmapDataPlanes:(unsigned char**)planes
    pixelsWide:((int)width pixelsHigh:(int) height
    bitsPerSample:(int)bps samplesPerPixel:(int)spp
    hasAlpha:(BOOL)alpha
    isPlanar:(BOOL)planar
    colorSpaceName:(NSString*)colorSpaceName
    bytesPerRow:(int)rowBytes bitsPerPixel:(int)pixelBits
```

RGB data will have three or four planes (without or with alpha) and CMYK will have four or five. White or black color space (grey-scales with 1.0 == white or 1.0 == black, respectively) will have one or two. If isPlanar is NO then only the first plane of planes will be read. This effectively places the rep in meshed mode.

Colorspace Names

```
  NSCalibratedWhiteColorSpace
  NSCalibratedBlackColorSpace
  NSCalibratedRGBColorSpace
  NSDeviceWhiteColorSpace
  NSDeviceBlackColorSpace
  NSDeviceRGBColorSpace
  NSDeviceCMYKColorSpace
  NSNamedColorSpace
  NSCustomColorSpace
 -(unsigned char*)bitmapData
```

Returns a pointer to the bitmap data or first plane if in planar configuration.

```
-(void)getBitmapDataPlanes:(unsigned char**)planes
```

The planes pointer should be an array of five character pointers. If the bitmap data is in planar configuration, each pointer will be initialized to point to one of the data planes. If there are less than five planes, the remaining pointers will be set to NULL.

```
-(NSData*)TIFFRepresentation
```

Returns a TIFF representation of the data, using the compression that is returned by getCompression:factor:. An

NSTIFFException or NSBadBitmapParameters-Exception may be raised if an error is encountered.

Logical Steps for Using an NSImage and NSBitmapImageRep

1) Define data buffer. See **Figure 4**.

```
  Example: unsigned char* dataBuffer =
     (void*)calloc((pixWide*pixHigh*bytesPerPix),
       sizeof(char));
2) Allocate and initialize an NSBitmapImageRep. See Figure 4.
  Example: myNSBitmapImageRep = [NSBitmapImageRep alloc];
     [myNSBitmapImageRep
       initWithBitmapDataPlanes:&dataBuffer
       pixelsWide:pixWide
       pixelsHigh:pixHigh
       bitsPerSample:SAMPLE_SIZE
       samplesPerPixel:SAMPLE_NUMBER hasAlpha:NO
       isPlanar:NO colorSpace:NSCalibratedRGBColorSpace
       bytesPerRow:(PIXEL_BYTES * pixWidth)
       bitsPerPixel:(PIXEL_BYTES * 8)];
```

The bitsPerSample argument describes the number of bits per color component. In the example code to follow, this is eight bits or one byte. To allocate three bytes per pixel one would thus set bitsPerSample to eight and samplesPerPixel to three.

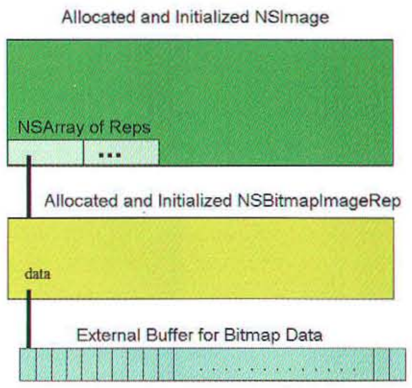

Allocated and Initialized NSImage
NSArray of Reps
...
Allocated and Initialized NSBitmapImageRep
data
External Buffer for Bitmap Data

Figure 4. *Relationship between NSImage, NSBitmapImageRep, and data buffer.*

3) Allocate and initialize an NSImage. Figure 4 describes the relationship between the NSImage and its representations.
Example:

```
myNSImage = [[NSImage alloc] init];
```

4) Tell the NSImage instance what representations to use.
Example:

```
[myNSImage addRepresentation:myBitmapImageRep];
```

5) Set the size to match the size of the image in the bitmap representation.
Example:

```
[myNSImage setSize:viewRectangle.size];//NSRect struct
```

6) Flip the y-axis. This is necessary, to make a right-handed coordinate system where (0,0) is the lower-left corner.
Example:

```
[myNSImage setFlipped];
```

7) Fill the buffer with data. Often in rendering, the color values will be a float or double triplet or quartet, with each element having values between 0.0 and 1.0. It is necessary to massage this raw data into the appropriate format for the data buffer. In the example code this means simply translating to a triplet of integers having values between 0 and 255.
Example:

```
color[i] *= 255; //Convert to unsigned int byte: Step 1
  if (modf(color[i],&theIntPart) >= 0.5)//Round to nearest
            //int: Step 2
  theIntColor[i] = theIntPart + 1;  //Assign to type int
            //var: Step 3a
  else
  theIntColor[i] = theIntPart;      //Step 3b
```

8: Draw the image to the NSView with either the compositeToPoint:operation: or dissolveToPoint:fraction: methods. One caveat is that if execution is not in the drawRect: method of a particular NSView when this drawing is performed, then focus needs to be locked on a particular NSView or NSImage fiirst. An example of this would be the case where one wants some initial startup images to appear in the application's NSViews to dazzle the users.

The drawRect: method is actually called during startup so one could still use it with some code to determine whether it was called during startup or called by the user. But... maybe information is needed from another object before doing the drawing... and since there is no guarantee as to which order objects inside the .nib file will be initialized, one cannot be certain other objects will exist yet when the NSView is being initialized. What to do?

Use a *delegate* of NSApp that automatically gets fired off after all objects get initialized upon startup. This delegate method would then lockFocus on the NSView to which it wants to draw, and have its NSImage object do a composite operation. The composite operation needs to know where to send the output. If the focus is locked, then unlock it after the drawing (unless an exception is raised).

Example:

```
[myNSImage compositeToPoint:theOriginPt
  operation:NSCompositeCopy];
[myNXImage dissolveToPoint:theOriginPt fraction:delta];
```

When and How to Use This Code Fragment

The following code fragment has been extracted from *Physics Lab's* RayTracerRenderer class. Relevant instance variables and global definitions are included with it. The initWithFrame: initializer is included but code has been removed from it that is not relevant to bitmaps and would complicate and obfuscate its purpose. Every effort has been made to provide a piece of code that can be hooked into your own project.

The drawRect: method is really the heart of the renderer. Several other support methods that are called are included after it, and listed in the order they are called. When an NSView receives a draw message it generates its own drawRect: message; the programmer does not explicitly send this message. What does this code do? A tiny overview of graphics techniques and terminology is in order.

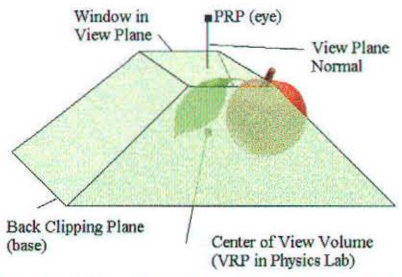
Physics Lab implements a mathematical model of various types of objects and their orientation in space. So mathematically there is a virtual four-dimensional Universe that is being modeled (the objects can move through three dimensions with respect to time). So there is the problem of how to display these three-dimensional objects at a given time, on a two-dimensional surface (the computer screen). It is true that the model is a virtual one, but it closely parallels the same process found in everyday life when one makes a movie of an actual three-dimensional scene. A mapping occurs from the four-dimensional world onto the rectangular, two-dimensional film.

The first thing the app does is it positions the *view volume* at the desired location in space. The user has to specify what part of the virtual mathematical Universe is of interest for mapping onto the screen. The view volume as used here is a truncated pyramidal shape with the flat "top" surface being the *projection surface* (also called the *window in the view plane*) for the objects contained within the volume.

Imagine a large, truncated, transparent pyramid (having a flat top surface rather than a point) with a pole sticking up out of the middle and long enough so that the end of the pole would be at the point of the pyramid if the pyramid had a point. The tip of the pole thus represents where the viewer's eye is located — the *Perspective Reference Point* or PRP. Imagine the dimensions of the top surface of the pyramid are let's say 40' x 30', to make a concrete example. There are user interface controls that allow the specification of the size and location of this view volume in space. See **Figure 5**.

Now, you want to draw what you see from your precarious vantage point atop the pole, into an NSView using the bitmaps we have been discussing. You know the NSView has let's say 150 x 200 pixels, so you will make a data buffer for the NSBitmapImageRep large enough to hold ninety-thousand bytes (for the RGB color model with no alpha: 150 * 200 * 3 = 90,000). So you need to look at some thirty-thousand equally spaced points on that top surface, which acts as a projection surface for the objects embedded inside the transparent pyramid.

The idea is to look at each of these points on the projection surface, and record the color that is seen there. Then set the color of the corresponding pixel on the computer screen to match that color. In effect, this is the mapping that the computer performs from the virtual 3D world to the computer screen for any given time t. It is important to calculate the spacing for a uniform grid to place on the top surface. Imagine looking down from the top of the pole at each of the points on the grid. If you see an object behind that point, you note exactly what the color is. If no object, then you set the color to the background color. The values you collect are the values that will be used for setting individual pixels. Our bitmaps. In a few paragraphs, that sums up the act of rendering in this app. Note that to get into canonical descriptions of view volumes and the like would require much more rigour because for example, the "pole" does not have to be in the center of the pyramid.


Window in View Plane
PRP (eye)
View Plane Normal
Back Clipping Plane (base)
Center of View Volume (VRP in Physics Lab)

Example of view volume with partially clipped object (apple). The top surface acts as a projection window and is mapped to a view port then to the display screen.

Figure 5. *View Volume Description.*

This code provides the functionality of creating the grid for the projection surface. The idea is to generate enough equally spaced grid points to collect exactly the amount of data to fill the NSView. You provide the way of generating a 2D surface that has the image data projected onto it, and this code will sample that data in a fair fashion to create and populate an NSBitmapImageRep of any given resolution — then draw that data into an NSView.

This code may be divided into several sections. Note that there is a dependency coded into the drawRect: method. It assumes that the UIInfoFields and UIPreferencesInfo-Fields pointers have already been allocated and defined if a rendering operation is in process. Since drawRect: is called when the NSView is being initialized, a test must be made to see whether initialization is occurring or whether an actual rendering operation is in progress. If the pointers are still set to NULL (thanks to the initWithFrame: method) then processing of basically the entire drawRect: method is avoided. There is conditional code in this snippet for checking return codes for error conditions. A cleaner approach would be to implement Rhapsody's assertion and exception handling mechanisms; this will probably be done in the next pass through the code.

Section 1

This is simply the relevant global definitions, and instance variables from the RayTracerRenderer class. These can be placed in the custom app with the appropriate name changes.

Section 2

The variable declarations in the drawRect: method are straightforward and commented. It probably will not be necessary to modify any of these, except for changing myName to match the name of your class.

Section 3

Initialization of the method variables should not require much modification either. The erase method is called to clear the view to a preset color. It uses a simple pswraps function called PSsetgray().

Section 4

This section calculates two different coordinate systems, plus the matrices necessary to move back and forth between them. Please note that in the calcCoordinateSystems method there are messages for other functions not included in this article. There are several reasons why this is so: 1) The methods require a knowledge of linear algebra; 2) To include them would more than double the amount of code to digest; 3) They reduce the focus of the article; 4) With a few simplifying assumptions described below, this whole section on creating coordinate systems is not even necessary.

First, the questions must be answered, "What are coordinate systems and why do we ever need them?" Most programmers are familiar with a screen coordinate system. In Rhapsody, the lower-left corner of the screen has the coordinates (0,0). Each window has its own coordinate system starting with (0,0) in its lower-left corner too. An NSView naturally has its own coordinate system implemented in the same way. Thus a hierarchy of coordinate systems exist wherein an image is positioned in an NSView which is positioned in an NSWindow which is positioned in an NSApp which is positioned on the screen. Of course a given window can have numerous NSViews and there are many more factors which shall be ignored for the sake of simplicity here. All of these various coordinate systems belong to the same group—they are associated with finding a specific pixel on the computer screen. This coordinate system is called UVN within this code. For graphics work, other coordinate systems must be considered that have nothing to do with the computer screen.

This program implements a spatial mathematical model of the Universe. A coordinate system is needed to describe objects using an arbitrary distance scale of angstroms, feet, meters, kilometers, parsecs, light-years, galactic radii or what-have-you. The literature often calls this the *World Coordinate System* or WCS. It is the coordinate system of the physical world volume one wishes to model.

Another coordinate system is used to describe the space within the view volume (which is positioned in space within WCS coordinates) as it faces the origin of the WCS. The view volume has its own (0,0,0) origin point for associating the relative position of every point within it. This coordinate system is called UVN1 within the code. Instead of X,Y,Z axes, there are U,V,N axes. Yet another coordinate system is used within the view volume if it is rotated so that it no longer faces the WCS origin. This system is called UVN2 within the code.

So, it is necessary to map a point described in WCS coordinates, to UVN1, then to UVN2, and finally to screen coordinates (UVN). It probably would be more intuitive to rename them and move from WCS to UVN2 to UVN1 and finally to UVN, right? This change may be implemented in the future. Fortunately some simplifications can be made so that it is only necessary to map from WCS straight to UVN screen coordinates, which should make this all easier to understand.

Here are the simplifying assumptions. Refer back to **Figure 5**. This view volume can be of any height, width, and depth. That is not a problem. The problem is if one wants to position it *anywhere*, and with any *aspect* or angle relative to the WCS axes. If one wants to do that, then one must implement multiple coordinate systems. On the other hand, much can be done with a fixed view volume. By fixed, it is meant that the coordinate system of the view volume exactly aligns with the World Coordinate System (which describes the physical Universe being modeled). The *Viewer's Reference Point* or VRP is the origin of the view volume coordinate system. It thus must equal (0,0,0) in WCS. This simplification requires the view volume to be centered at (0,0,0) in the physical Universe being modeled. It cannot be located at any other position. The objects must be located in the view volume in order to be displayed — so they must be modeled near the origin.

Further, the window in the view plane (top surface of the pyramid) is parallel to the plane created by the X-Y axes. The simplest approach is to further have the "eye" or Perspective Reference Point (PRP) located on the positive Z axis, and have the VRP located at the center of the view volume. For mapping purposes the +Z axis can be thought of as coming "out" of the monitor, and -Z "going into" the monitor. In a standard way, +X goes to the right and +Y goes upward. This preserves a right-handed coordinate system. The Z coordinate for the window thus becomes frontN and the Z coordinate for the back clipping plane (a negative Z value) becomes rearN — two instance variables in the RayTracerRenderer class.

The scale must be the same in the screen coordinate system and WCS.

To drive all this home, a simple example is in order. Imagine that the units of distance are meters. The window in the view plane is two meters wide and two meters high. It is located at N = 1, and it is parallel to the plane formed by the U-V axes. WCS is in meters and UVN is in meters. Their axes exactly align with one another. WCS exactly equals UVN. The PRP is located at N = 2 and the back clipping plane is located at N = -1. The origin or VRP of the view volume is located at (0,0,0). Any objects that fall within the truncated pyramid formed by this view volume will be visible. The view volume can not be rotated or scaled or translated away from its origin point. Life is good.

With these simplifications it is possible to do all calculations in one coordinate system (WCS). No translations, scalings, and rotations are required to move from one to another. Actually this is technically not quite true for a ray tracer as one generally *still* must convert the cast ray to a "generic object" coordinate system to see if it intersects with the graphics primitives. But, as far as this code snippet is concerned, no changes in coordinate systems would be required. The "generic object" coordinate system is found in the RayTracerShader instance that is messaged by drawRect: to calculate the actual pixel color — and that code is not included here. If another type of shader is used, that coordinate system would not be needed.

Section 5

This is the meaty part. The nested for-loops perform the mapping from the window in the view plane to a mathematical viewport, which has a one-to-one correspondence with each pixel.

Some simple math is performed to derive deltaU and deltaV. These are the increments that for the window of given dimensions, will generate the same amount of points as there are pixels in the NSView image. The actual for-loops use row and col as indices. These refer to the NSView image and as can be seen from **Figure 6**, use a different coordinate system than that of the window. A cast ray is created, originating at the "eye" or PRP and passing through a point on the window defined by uPixel and vPixel.

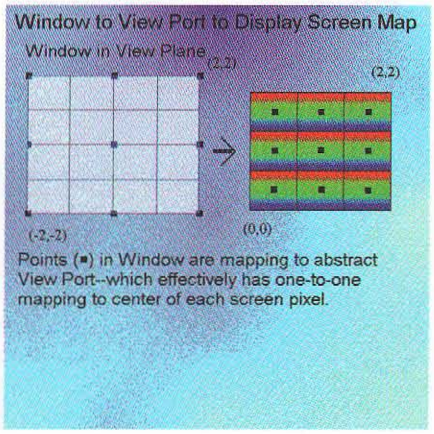

Window to View Port to Display Screen Map
Window in View Plane
(2,2)
(2,2)
(-2,-2)
(0,0)
Points (▪) in Window are mapping to abstract View Port--which effectively has one-to-one mapping to center of each screen pixel.

Figure 6. *Mapping from window in view plane to screen.*

This ray is passed to the shader. Many types of shaders are possible. *Physics Lab* uses a recursive ray tracing algorithm (the nested for-loops are actually part of it). For this article, the message to the shader is a black box that simply returns the appropriate color for a given pixel. Note that the NS3.3 development environment contains an entire suite of powerful 3D rendering classes in 3DKit. This was not used here for a couple of reasons, mainly because the current version at the time of development did not allow one to hook in a custom shader, but also for portability reasons.

Once the color value is returned, it is converted from the range 0.0 - 1.0 to the range 0 - 255. Further, it is rounded to the nearest integer. That is because the image data uses single bytes for each color component or sample. The data buffer is populated by the time the nested for-loops complete, and then a message to compositeToPoint:operation: in the NSImage completes the rendering process. Since all of this is happening inside of the NSView's own drawSelf:: method, it is not necessary to lock focus. Code is included for rendering the image using pswraps also. This may be useful for performance comparison purposes.

CODE SNIPPET:

```
//////////////////////SECTION 1: STRUCTURES & INSTANCE VARS////////////////////////
typedef struct _GL_Ray
{
   double start[3];
   double dir[3];
   double normalizedDir[3];
   double mediumRefractivity; //Index of refraction for
                              //medium ray traveling
                              //through. Useful for
                              //Constructive Solid
                              //Geometry or anytime have
                              //compound object boundaries.
if == 0, then a vacuum.
   double length;
} GL_Ray;
typedef struct _GL_UIInfo
{
   double RAFieid;        //Right Ascension from user interface.
   double DField;         //Declination from user interface.
   double VVRAField;      //View Volume Right Ascension from UI
   double VVDField;       //View Volume Declination from UI
   double CTField;        //Current Time from UI
   double TBIField;       //Time Between Images  from UI
   double FITField;       //First Image Time from UI
   double DFCField;       //Distance From Center from UI
   int VPWidthField;      //View Plane Width from UI
   int VPHeightField; //View Plane Height from UI
   int VVDepthField;      //View Volume Depth from UI

   id OList; //Object array.
   id LList; //Light array.
} GL_UIInfo;
typedef struct _GL_UIPreferencesInfo
{
   double ambientField[3];   //Ambient light RGB array.
   int PRPDistanceField; //Distance from view plane.

   //Diffuse coefficients
   double attenuationFactorC1Field;    //Light-source
                                       //attenuation factor.
   double attenuationFactorC2Field; //Light-source
                                    //attenuation factor.
   double attenuationFactorC3Field; //Light-source
                                    //attenuation factor.

   //Global brightness controls
   double ambientBrightnessField;   //These settings scale
                                    //final color.
   double diffuseBrightnessField;   //Brightness for all objs.
                                    //objects in the scene.
   double specularBrightnessField; //(range 0 - 1).

   int maxRayTreeDepthField; //How recursive you want to be?
   int statusMode; //TERSE, VERBOSE, or DEBUGGING?

} GL_UIPreferencesInfo;

//Global #defines
#define GL_SAMPLE_SIZE 8 //Number of bits in a sample (one
                         //color component).
#define GL_SAMPLE_NUMBER 3 //Number of samples per pixel.
#define GL_PIXEL_BYTES 3   //May be different from
                           //GL_SAMPLE_NUMBER *
                           //GL_SAMPLE_SIZE / 8 if
                           //GL_SAMPLE_SIZE is non-integer
                           //multiple of a byte and data
                           //aligned along word boundary.

///INSTANCE VARIABLES FOR RENDERER (USED IN THIS CODE FRAGMENT)//
double PRP[3]; //Projection Reference Point (eye).
double VRP[3]; //In XYZ (WCS) coords.
double UVN1[3][3],UVN2[3][3]; //Axes defined in XYZ,
                              //UVN1 coords, respectively.

double inverseUVN1[3][3];//Inverse matrices
double inverseUVN2[3][3];//for moving betw coord systems.

double VUP1[3],VUP2[3];        //Defines "which way is up" in
                               //containing coord
                               //systems. (XYZ, UVN1 coords,
                               //respectively)

GL_UIInfo* UIInfoFields;       //Instance of structure for
                               //containing UI data from main
                               //window and orientation window.

GL_UIPreferencesInfo* UIPreferencesInfoFields; //Struct
                                               //containing info
                                               //from preferences panel.

int frontN, rearN; //Z coords (if view volume fixed) or N
                   //coordinates in UVN screen coordinate system, of
                   //where the front and rear clipping planes of the
                   //view volume are located. Note front clipping
                   //plane contains view window.
```

All the signs point to Developer Depot...

And, you know the address!
http://www.devdepot.com

http://www.scientific.com

Professional Software Developers

Looking for career opportunities?
Check out our website!

Nationwide Service
Employment Assistance
Resume Help
Marketability Assessment
Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)
das@scientific.com

BMS

ATTORNEY AT LAW

WILLS & TRUSTS, INTELLECTUAL PROPERTY, CORPORATE, COMMERCIAL AND CONTRACT LAW

California Lawyer focusing on Corporate, Commercial and Contract law, as well as Intellectual Property and Wills & Trusts.

If you need legal assistance in trying to start up or maintain a business, or if you need help securing copyright and trademark protection for your valuable work, please give me a call or an e-mail. Reasonable fees.

Visa • MC • American Express • Discover

BMS

(310) 553-4054 • brad@sniderman.com • http://www.sniderman.com

```
int aWindow[4], viewport[4];        //LL and UR corners for view
                                    //window and view port.

////////////////////////////////END OF INSTANCE VARIABLES////////////////////////////////

//////////////////////////////////////END OF SECTION 1//////////////////////////////////////

//The drawRect: method is called indirectly. The application code sends a display msg
//to an NSView and the NSView eventually sends itself a drawRect: msg, executing this
//code. In essence, "rays" are fired through the points in space that map to the pixels on
//the screen. This code determines the mappings and creates the rays. It then sends the
//rays to a "black box" that returns the color of any object (if any) that was "struck" by
//the ray as it traces a path through its mathematical Universe.

////////////////////////////////////////////////////////////////////////////////////////////

-drawRect:(NSRect *)r
{

//////////////////////////////////SECTION 2: DECLARATIONS//////////////////////////////////

  //Used for status msgs
  char myName[] = "[rayTracerRenderer drawRect]";
  int status = GOOD;

  int i, row, col, theByteOffset; //Various indexes.

  double color[3];         //The raw shader color data ranging
                           //from 0.0 to 1.0.
  double theIntPart;       //Used in intermediate step for
                           //converting data from float to int.
  int theIntColor[3];      //Integer value of color returned
                           //from shader.

  double aGray;            //Contains color value in PSWraps code
                           //(commented out).

  //View window variables.
  double deltaU, deltaV, uPixel, vPixel, targetPt[3];

  GL_Ray newRay;           //Ray from PRP to (uPixel, vPixel,
                           //nFront)

  unsigned char* bitmapData = NULL; //Buffer for image data

  NSPoint theOriginPt;       //Used to tell NSimage where in
                             //NSView to begin
                             //compositing (contains lower-left
                             //point of the image in NSView
                             //coordinate system)

  NSRect theRect, viewRect;//Rectangle structures.

  NSImage theImage;
  NSBitmapImageRep theBitmapRep;   //OUR BITMAP!

///////////////////////////END OF SECTION 2: DECLARATIONS///////////////////////////

////////////////////////////////SECTION 3: INITIALIZE////////////////////////////////////////

  //Get view bounds
  viewRect = [self bounds];

  //Allocate memory for buffer, based on size of NSView bounds rectangle
  bitmapData = (void*)calloc(viewRect.size.width *
viewRect.size.height * GL_PIXEL_BYTES,sizeof(unsigned
char));

  if (bitmapData==NULL)
    status = BAD;

  //Initialize originPt
  theOriginPt.x = 0.0;
  theOriginPt.y = 0.0;

  //Check and see if render button has been pressed or if this method
  //called simply by initialization display msg at startup time
  if (status==GOOD)
  {
    //These will have been initialized by time execution gets here
    //if Render button has been pressed. Both SHOULD be NULL if it is
    //the init call, but using OR as it is an abort condition if EITHER
    //is NULL.
    if ((UIInfoFields==NULL) ||
      (UIPreferencesInfoFields==NULL))
    {
      [self erase]; //If called during init, clear NSView
      status = BAD; //but don't do any other processing in
                    //this method.
    }
  }
  else
    [self erase]; //If status bad, still clear the View;
                  //don't do any
                  //other processing in this method though.

////////////////////////////END OF SECTION 3: INITIALIZE////////////////////////////////

//////////////////////SECTION 4: GENERATE COORDINATE SYSTEMS///////////////////

//NOTE: This section is not necessary if you make the following assumptions:
// 1) Your view volume has an origin that is at (0,0,0) in the World
//   Coordinate System (X,Y,Z coordinates).
// 2) The axes of your view volume are parallel to the X,Y,Z axes.
// 3) The scale of your view volume is equivalent to that of the X,Y,Z
//   coordinate system.
// 4) The view plane is parallel to the X-Y plane and offset in positive Z dir
// Put another way: Your U,V,N screen coord system is exactly the same as the
// X,Y,Z World Coordinate System.

  if (status==GOOD)
    status = [self calcCoordinateSystems];

///////////////END OF SECTION 4: GENERATE COORDINATE SYSTEMS/////////////////

////////////////////////////////SECTION 5: RAY CASTING//////////////////////////////////////

/*******PERFORM RAY CASTING IN WCS (X,Y,Z)**********************/

  if (status==GOOD)
  {
    //Convert eye from second uvn to xyz. PRP DFC provided by pref panel
    //Uses UVN2, UVN1, and VRP instance variables.
    //PRP = Distance From Center (of view plane) plus half of view
    //volume depth.
    //Not necessary for fixed view volume.
    status = [self convertUVN2toXYZ:PRP];
  }

  if (status==GOOD)
  {
    //Calc viewport iteration factor from view window to viewport
    //Uses aWindow[], and viewport[] instance variables which are
    //set by getUIInfo. Note that window size and viewport size
    //forced by getUI method to be of even size (thus always has
    //exact center). A 200 x 100 window has coords (-100,-50,100,50)
    //and a 50 x 50 viewport has coords (0,0,49,49).

    deltaU = ((float)aWindow[2] - (float)aWindow[0]) /
      ((float)pixelsWide - 1);
    deltaV = ((float)aWindow[3] - (float)aWindow[1]) /
      ((float)pixelsHigh - 1);
  }

  if (status==GOOD)
  {
    vPixel = aWindow[1]; //Window bottom.

    //From bottom to top.
    for (row = 0; row <= (pixelsHigh - 1); row++)
{
      uPixel = aWindow[0]; //Window left.
      //From left to right.
      for (col = 0; col <= (pixelsWide - 1); col++)
      {
        //Construct ray from eye through (u,v,frontN)
        //By first making targetPt array in UVN2 coords.
        targetPt[0] = uPixel;
        targetPt[1] = vPixel;
        targetPt[2] = frontN;
```

```
//Convert targetPt to XYZ coords
//Uses UVN2, UVN1, and VRP instance variables.
//Not necessary if your UVN==XYZ

status = [self convertUVN2toXYZ:targetPt];
if (status != GOOD)
  break;

//Calc length and normalizedDir elements of ray struct
//Returns nil if length of ray is zero (cannot normalize).
//This would only happen if PRP distance from view plane
//is zero. NOTE: PRP DFC field validation should not
//allow value of zero or neg. (PRP has to be in front of
//view plane).

if ([talkToMathUtilities defineRay:&newRay
  withStart:PRP andTargetPt:targetPt]==nil)
{
  status = BAD;
  break;
}

//TraceRay has to return status since recursively
//used for intra-object communication too.
//This is a Black Box for the purposes of this article.
//Suffice that for the given ray, an appropriate
//pixel color is returned. Implement any algorithm you
//desire here. The basic idea is to extend this
//mathematical ray and find what objects if any it
//intersects within the view volume. If no intersection
//then return ambient light color as the background.
//If intersection then at point of intersection of
//nearest object determine the color and brightness
//based on the object and all light sources in the scene
//(diffuse, specular, plus recursive reflective and
//refractive rays). Easy as that.

status = [talkToShader traceRay:newRay atDepth:1
  returnsColor:color];
if (status!=GOOD)
  break;

for (i = 0; i < 3; i++)
{
  //Clean up and insure an invariant on domain of data
  //(in case of bugs in your shader)

  //Clamp color to max of 1.0
  if (color[i] > 1.0)
    color[i] = 1.0;
  //and min of 0.0
  if (color[i] < 0.0)
    color[i] = 0.0;

  //COMMENT THIS OUT if want to use PSWraps code

  color[i] *= 255; //Convert to unsigned int byte
  if (modf(color[i],&intPart) >= 0.5)
    intColor[i] = intPart + 1;
  else
    intColor[i] = intPart;
  //TO HERE

}

/*      //USE THIS CODE if want to see pswraps version
        //PS monochrome
        aGray = (color[0] + color[1] + color[2]) / 3.0;
        PSsetgray(aGray); //Normalized total
        PSCompositeRect(col,row,1,1,NSCompositeCopy);
        //TO HERE
*/
        //Use this code if want to see NSImage version
        //Calculate starting byte index in data array
        theByteOffset = GL_PIXEL_BYTES * ((row *
          viewRect.size.width) + col);
        for (i = 0; i < 3; i++)
          bitmapData[byteOffset + i] = intColor[i];
        //TO HERE
```

```
        uPixel += deltaU;
      }//End of inner-for
      vPixel += deltaV;
      if (status!=GOOD)
        break;
    }//End of outer-for

    //COMMENT OUT THIS CODE if want to use PSWraps version

    if (theImage!=nil)
    {
      [theImage release]; //Note releasing NSImage
      //releases managed reps.
      theImage = nil;
    }


    theImage = [[NSImage alloc]
      initWithSize:viewRect.size]; //No retain
    if (theImage==nil)
      status = BAD;
    else
    {
      [theImage setFlipped:YES]; //Flip coords so (0,0) is
                    //upper left
                    //instead of lower left corner.
                    //(May not need this depending on order
                    //in which your data is generated).
    }

    //Initialize NSBitMapImageRep
    theBitmapRep = [NSBitmapImageRep alloc]; //No retain
    if (theBitmapRep==nil)
      status = BAD;
    else
    {
      //Note ampersand
      [theBitmapRep initWithBitmapDataPlanes:&bitmapData
        pixelsWide:viewRect.size.width
        pixelsHigh:viewRect.size.height
        bitsPerSample:GL_SAMPLE_SIZE
        samplesPerPixel:GL_SAMPLE_NUMBER hasAlpha:NO
        isPlanar:NO
colorSpaceName:NSCalibratedRGBColorSpace
        bytesPerRow:(viewRect.size.width * GL_PIXEL_BYTES)
        bitsPerPixel:(GL_PIXEL_BYTES * 8)];
      [theImage addRepresentation:theBitmapRep] ;
    }


    //Composite the NSImage to the NSView
    [GL_NSImage compositeToPoint: theOriginPt
      operation:NSCompositeCopy];

    //TO HERE

  }

  if (status != GOOD)
    return nil;

  return self;
}
///////////////////////////END OF SECTION 5: RAY CASTING///////////////////////////////

- (void)erase //Clears the view
{
  PSsetgray(NSWhite);
  NSRectFill([self bounds]);
  return;
}

/////////////////////////////////////////////////////////////////////////////////////////
//The calcCoordinateSystems method is another black box here. If some simplifying
//assumptions are made as described in the article, this entire section is not needed.
//It calls functions that are not included in this article.
/////////////////////////////////////////////////////////////////////////////////////////

- (BOOL)calcCoordinateSystems
{
  int status = GOOD;
```

```
/*CREATE SECOND VIEWER REFERENCE COORD FROM WORLD COORD*/

if (status==GOOD)
{
   //Calculate VRP based on UI specification for right ascension,
   //declination, and distance from center

   status = [self calcVRPusingRAandDandDFC]; //In WCS
}

if (status==GOOD)
{
   //Get U,V, and N axes for unrotated viewing volume
   //Status should always be GOOD as meth terminates app on error
   //Calculates uNormal,vNormal,nNormal

   status = [self calcUVN1usingVRP];
}

if (status==GOOD)
{
   //Get U,V, and N axes for ROTATED viewing volume.
   //Calcs uNormal, vNormal, nNormal
   status = [self calcUVN2byRotUVN1usingVVRAandVVD];
}
/****************************************************************/
}
```

Summary

This article has mostly focused on one of the four NSImageRep subclasses plus NSView and NSImage. These classes have a tremendous number of uses and possible configurations that go beyond the scope of this article. The performance figures will vary from app to app of course because it really depends where the app is spending its time. In the test application a large percentage of the time is spent in shading calculations, yet the speed increase is still roughly twice that over using pswraps to draw individual pixels. This test was done with one sphere object and one point light. Adding objects to the scene reduced this performance gain as more and more time is consumed computationally by calculating intersections, etc.

It is likely that if one measured the time solely used for drawing the pixels to screen (and ignoring time spent creating the pixel data) that the performance differential would be much greater. I suspect the performance increase is partly obscured in this case because of the relatively large amount of time spent generating the data. Here is an example of how this could happen. Suppose that ten seconds are required to generate the pixel data for the test rendering. Now imagine that while using pswraps a rendering takes a total of twenty seconds. While using bitmaps the same rendering takes a total of eleven seconds. The test would suggest a speed increase of roughly 2X when in reality the speed increase is more on the order of 10X (it took one second instead of ten to draw the image data to the screen). The pswraps code is included in the snippet (commented out) for those who wish to run formal comparison tests using a profiler.

The *Physics Lab* software is an ongoing project aimed at visualization of field phenomena such as gravity, electromagnetism, and subatomic forces. More information about this product can be obtained from the web site at: <http://www.gj.net/prv>.

MT

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

LIST OF PRODUCTS

We'll show you the way to easy web publishing!

WebSTAR® 3

Now Includes

GoLive CyberStudio Personal Edition
(a $99 value!)

Order online at www.starnine.com
or call 800/525-2580
Email: sales@starnine.com

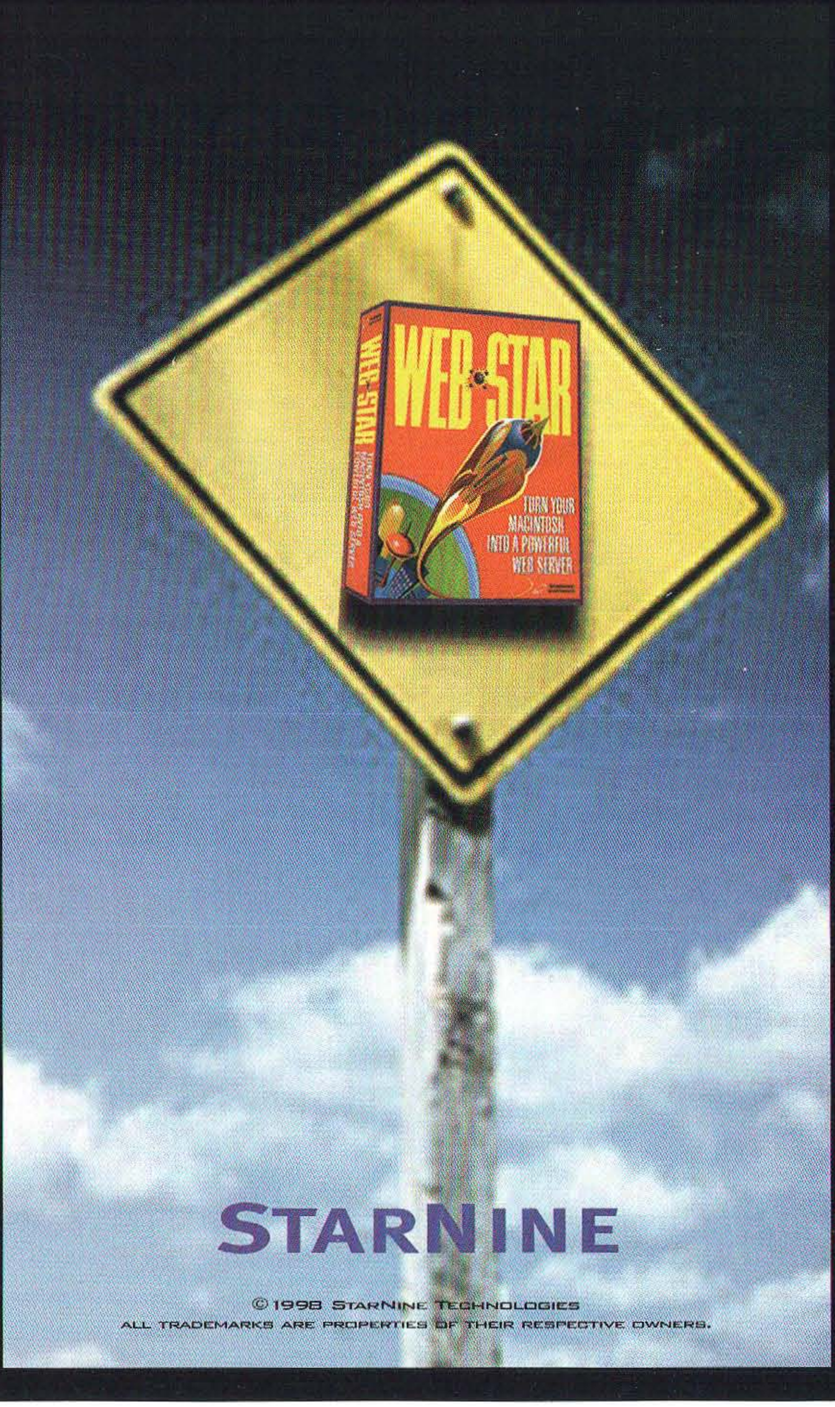
WebSTAR® The Preferred Route for Powerful Web Serving

Are you ready to serve your own Web site but don't know the best way? Just follow the thousands of satisfied Webmasters who have chosen WebSTAR as the fastest road to Web serving.

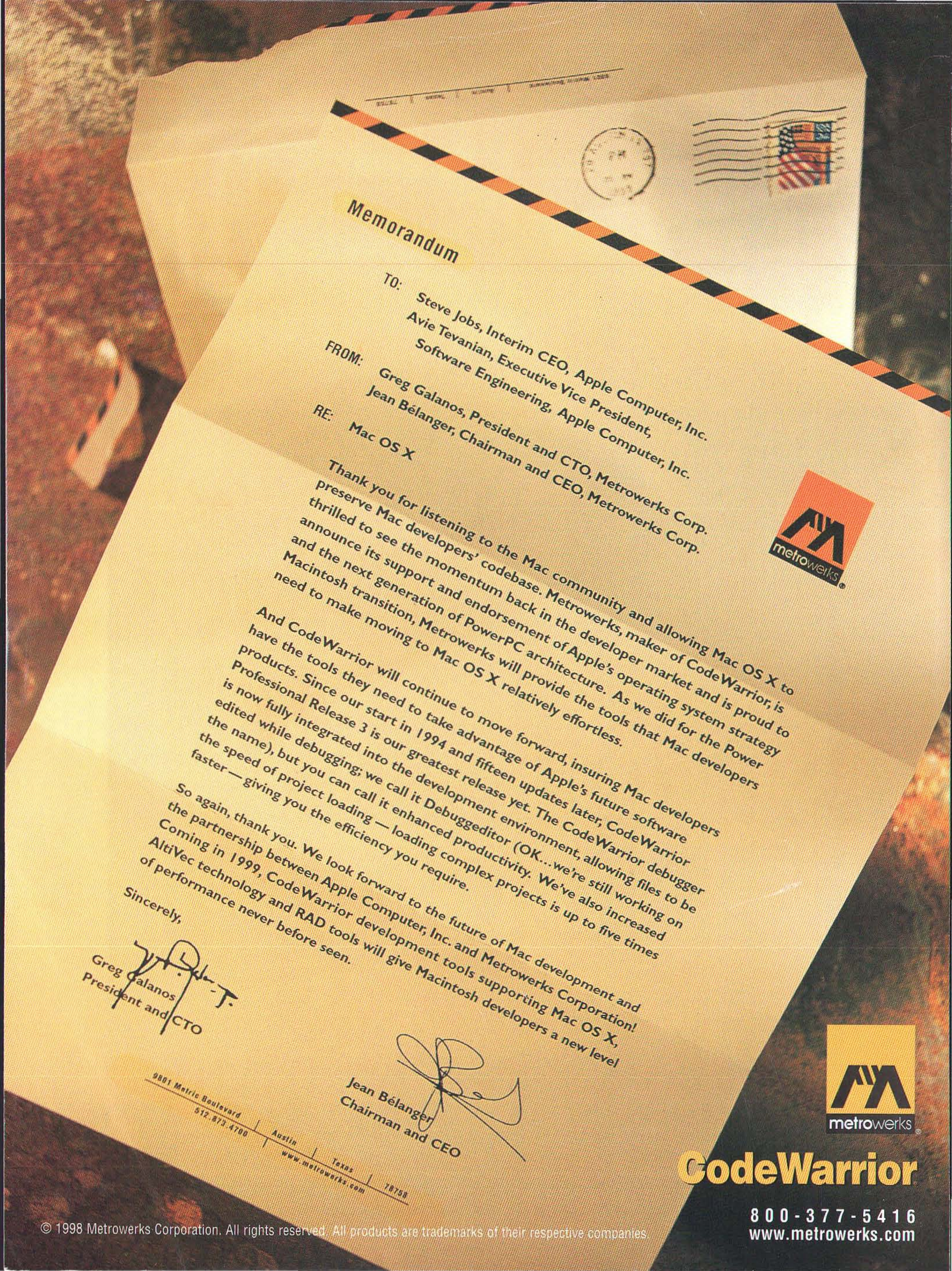
With WebSTAR you'll be on your way in minutes. Remote Administration, Server-Side Includes, image maps, password protection, encryption, logging and indexing are all included. And version 3 now includes FTP and proxy servers, search, and the ability to serve hundreds of sites from one server. Wondering about performance? WebSTAR is fast enough to serve over twenty million hits a day!

You can go even further by adding CGIs, Plug-ins and Java Servlets™. If you can imagine it, you can do it with WebSTAR! And rest assured you won't be alone on this road — StarNine offers top-flight technical support.

Directions: Stop by at www.starnine.com to download a full-working demo copy. Try it, then buy it — by ordering online or by calling us at 800/525-2580. You'll reach your destination sooner than you think!



STARNINE



Memorandum

TO: Steve Jobs, Interim CEO, Apple Computer, Inc.
Avie Tevanian, Executive Vice President,
Software Engineering, Apple Computer, Inc.

FROM: Greg Galanos, President and CTO, Metrowerks Corp.
Jean Bélanger, Chairman and CEO, Metrowerks Corp.

RE: Mac OS X

metrowerks

Thank you for listening to the Mac community and allowing Mac OS X to preserve Mac developers' codebase. Metrowerks, maker of CodeWarrior, is thrilled to see the momentum back in the developer market and is proud to announce its support and endorsement of Apple's operating system strategy and the next generation of PowerPC architecture. As we did for the Power Macintosh transition, Metrowerks will provide the tools that Mac developers need to make moving to Mac OS X relatively effortless.

And CodeWarrior will continue to move forward, insuring Mac developers have the tools they need to take advantage of Apple's future software products. Since our start in 1994 and fifteen updates later, CodeWarrior Professional Release 3 is our greatest release yet. The CodeWarrior debugger is now fully integrated into the development environment, allowing files to be edited while debugging; we call it Debuggeditor (OK...we're still working on the name), but you can call it enhanced productivity. We've also increased the speed of project loading — loading complex projects is up to five times faster — giving you the efficiency you require.

So again, thank you. We look forward to the future of Mac development and the partnership between Apple Computer, Inc. and Metrowerks Corporation! Coming in 1999, CodeWarrior development tools supporting Mac OS X, AltiVec technology and RAD tools will give Macintosh developers a new level of performance never before seen.

Sincerely,

[illegible]

Greg Galanos
President and CTO

[illegible]

Jean Bélanger
Chairman and CEO

9861 Metric Boulevard | Austin | Texas | 78758
512.873.4700 | www.metrowerks.com

metrowerks®

CodeWarrior

800-377-5416
www.metrowerks.com